

O'REILLY®

2-е издание

SQL

Сборник рецептов

Решения и методики
построения запросов
для разработчиков
баз данных



Энтони Молинаро
Роберт де Грааф



SECOND EDITION

SQL Cookbook

*Query Solutions and Techniques
for All SQL Users*

Anthony Molinaro and Robert de Graaf

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY

**Энтони Молинаро
Роберт де Грааф**

SQL

Сборник рецептов

Решения и методики
построения запросов
для разработчиков
баз данных

2-е издание

Санкт-Петербург
«БХВ-Петербург»
2022

УДК 004.43
ББК 32.973.26-018.1
М75

Молиново, Э.

М75 SQL. Сборник рецептов. — 2-е изд.: Пер. с англ. / Э. Молиново, Р. де Грааф. — СПб.: БХВ-Петербург, 2022. — 592 с.: ил.

ISBN 978-5-9775-6759-6

Рассмотрены готовые рецепты для решения практических задач при работе с СУБД Oracle, DB2, SQL Server, MySQL и PostgreSQL. Описаны извлечение записей из таблиц, сортировка результатов запросов, принципы работы с несколькими таблицами, обработка запросов с метаданными. Рассказывается о способах поиска данных средствами SQL, о составлении отчетов и форматировании результирующих множеств, работе с иерархическими запросами. Рассматриваются использование оконных функций, обобщенных табличных выражений (OTB), сбор данных в блоки, формирование гистограмм, текущих сумм и подсумм, агрегация скользящего диапазона значений. Описан обход строки и ее синтаксический разбор на символы, приведены способы упрощения вычислений внутри строки. Во втором издании учтены все изменения в синтаксисе и архитектуре актуальных реализаций SQL.

*Для программистов, разработчиков
и администраторов баз данных*

УДК 004.43
ББК 32.973.26-018.1

Группа подготовки издания:

Руководитель проекта	<i>Павел Шалин</i>
Зав. редакцией	<i>Людмила Гауль</i>
Редактор	<i>Григорий Добин</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Оформление обложки	<i>Карины Соловьевой</i>

© 2021 BHV

Authorized Russian translation of the English edition of *SQL Cookbook 2nd edition* ISBN 9781492077442

© 2021 Robert de Graaf

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Авторизованный перевод с английского языка на русский издания *SQL Cookbook 2nd edition* ISBN 9781492077442

© 2021 Robert de Graaf

Перевод опубликован и продается с разрешения компании-правообладателя O'Reilly Media, Inc.

Подписано в печать 06.07.21.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 47,73.

Тираж 1500 экз. Заказ № 1728.

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Отпечатано с готового оригинал-макета

ООО "Принт-М", 142300, М.О., г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-1-492-07744-2 (англ.)

ISBN 978-5-9775-6759-6 (рус.)

© Robert de Graaf, 2021

© Перевод на русский язык, оформление.

ООО "БХВ-Петербург", ООО "БХВ", 2021

Оглавление

ПРЕДИСЛОВИЕ	30
Для кого предназначена эта книга?	31
Чего нет в этой книге?.....	31
Платформы и версии SQL.....	32
Таблицы, используемые в книге	32
Соглашения, используемые в книге.....	34
Типографские соглашения.....	34
Соглашения по написанию кода	34
Благодарности за второе издание.....	35
Благодарности за первое издание.....	36
ГЛАВА 1. ИЗВЛЕЧЕНИЕ ЗАПИСЕЙ	39
1.1. Извлечение из таблицы всех строк и столбцов	39
Задача.....	39
Решение	39
Обсуждение.....	39
1.2. Извлечение из таблицы подмножества строк	40
Задача.....	40
Решение	40
Обсуждение.....	40
1.3. Возвращение строк по нескольким условиям	40
Задача.....	40
Решение	40
Обсуждение.....	41
1.4. Извлечение из таблицы подмножества столбцов	41
Задача.....	41
Решение	41
Обсуждение.....	42
1.5. Задание столбцам значимых имен	42
Задача.....	42
Решение	42
Обсуждение.....	43
1.6. Обращение к столбцу в предикате <i>WHERE</i> по его псевдониму	43
Задача.....	43
Решение	43
Обсуждение.....	43
1.7. Конкатенация значений столбцов	44
Задача.....	44

Решение	44
DB2, Oracle, PostgreSQL	44
MySQL	45
SQL Server	45
Обсуждение	45
1.8. Использование условной логики в операторе <i>SELECT</i>	45
Задача	45
Решение	46
Обсуждение	46
1.9. Ограничение числа возвращаемых строк	46
Задача	46
Решение	46
DB2	46
MySQL и PostgreSQL	47
Oracle	47
SQL Server	47
Обсуждение	47
1.10. Извлечение из таблицы произвольных записей	48
Задача	48
Решение	48
DB2	48
MySQL	49
PostgreSQL	49
Oracle	49
SQL Server	49
Обсуждение	49
1.11. Поиск значений <i>NULL</i>	50
Задача	50
Решение	50
Обсуждение	50
1.12. Преобразование значений <i>NULL</i> в реальные значения	50
Задача	50
Решение	50
Обсуждение	50
1.13. Поиск по шаблону	51
Задача	51
Решение	52
Обсуждение	52
1.14. Подведем итоги	52
ГЛАВА 2. СОРТИРОВКА РЕЗУЛЬТАТОВ ЗАПРОСА	53
2.1. Возвращение результатов запроса в заданном порядке	53
Задача	53
Решение	53
Обсуждение	53
2.2. Сортировка по нескольким столбцам	54
Задача	54
Решение	55
Обсуждение	55

2.3. Сортировка по подстрокам.....	55
Задача.....	55
Решение	56
DB2, MySQL, Oracle и PostgreSQL	56
SQL Server	56
Обсуждение.....	56
2.4. Сортировка смешанных буквенно-цифровых данных	56
Задача.....	56
Решение	58
Oracle, SQL Server и PostgreSQL	58
DB2.....	58
MySQL	59
Обсуждение.....	59
2.5. Обработка значений <i>NULL</i> при сортировке.....	60
Задача.....	60
Решение	60
DB2, MySQL, PostgreSQL и SQL Server	61
Oracle	63
Обсуждение.....	65
2.6. Сортировка по ключу, зависящему от данных.....	66
Задача.....	66
Решение	66
Обсуждение.....	67
2.7. Подведем итоги	67
ГЛАВА 3. РАБОТА С НЕСКОЛЬКИМИ ТАБЛИЦАМИ	68
3.1. Размещение одного набора строк над другим	68
Задача.....	68
Решение	68
Обсуждение.....	69
3.2. Объединение взаимосвязанных строк.....	70
Задача.....	70
Решение	70
Обсуждение.....	70
3.3. Поиск строк с общими данными в двух таблицах	72
Задача.....	72
Решение	72
MySQL и SQL Server	73
DB2, Oracle и PostgreSQL	73
Обсуждение.....	73
3.4. Извлечение из одной таблицы значений, отсутствующих в другой	74
Задача.....	74
Решение	74
DB2, PostgreSQL и SQL Server	74
Oracle	74
MySQL	74
Обсуждение.....	75
DB2, PostgreSQL и SQL Server	75
Oracle	75
MySQL	75

3.5. Извлечение строк из таблицы, не соответствующих строкам в другой таблице	80
Задача.....	80
Решение	80
DB2, MySQL, PostgreSQL и SQL Server	80
Обсуждение.....	80
3.6. Добавление в запрос независимых объединений.....	81
Задача.....	81
Решение	83
Обсуждение.....	83
3.7. Проверка двух таблиц на идентичность	84
Задача.....	84
Решение	85
DB2 и PostgreSQL	85
Oracle	85
MySQL и SQL Server	86
Обсуждение.....	87
DB2, Oracle и PostgreSQL	88
MySQL и SQL Server	89
3.8. Выявление и устранение проблемы декартовых произведений.....	91
Задача.....	91
Решение	92
Обсуждение.....	92
3.9. Выполнение объединений при использовании агрегатных функций	93
Задача.....	93
Решение	95
MySQL и PostgreSQL	95
DB2, Oracle и SQL Server	95
Обсуждение.....	96
MySQL and PostgreSQL.....	96
DB2, Oracle и SQL Server	97
3.10. Выполнение внешних объединений при использовании агрегатных функций.....	97
Задача.....	97
Решение	99
DB2, MySQL, PostgreSQL и SQL Server	99
Обсуждение.....	100
3.11. Возвращение отсутствующих данных из нескольких таблиц.....	101
Задача.....	101
Решение	103
DB2, MySQL, PostgreSQL и SQL Server	103
Oracle	103
Обсуждение.....	103
3.12. Значения <i>NULL</i> в вычислениях и сравнениях	105
Задача.....	105
Решение	105
Обсуждение.....	105
3.13. Подведем итоги	106

ГЛАВА 4. ВСТАВКА, ОБНОВЛЕНИЕ И УДАЛЕНИЕ ЗАПИСЕЙ	107
4.1. Вставка новой записи	107
Задача.....	107
Решение	108
Обсуждение.....	108
4.2. Вставка значений по умолчанию	108
Задача.....	108
Решение	109
Обсуждение.....	109
4.3. Переопределение значения по умолчанию значением NULL	110
Задача.....	110
Решение	110
Обсуждение.....	110
4.4. Копирование строк одной таблицы в другую	110
Задача.....	110
Решение	111
Обсуждение.....	111
4.5. Копирование определения таблицы	111
Задача.....	111
Решение	111
DB2.....	111
Oracle, MySQL и PostgreSQL	111
SQL Server	112
Обсуждение.....	112
DB2.....	112
Oracle, MySQL и PostgreSQL	112
SQL Server	112
4.6. Вставка строк одновременно в несколько таблиц	112
Задача.....	112
Решение	113
Oracle	113
DB2.....	113
MySQL, PostgreSQL и SQL Server.....	114
Обсуждение.....	114
Oracle	114
DB2.....	114
MySQL, PostgreSQL и SQL Server.....	114
4.7. Блокировка вставки данных в определенные столбцы.....	114
Задача.....	114
Решение	114
Обсуждение.....	115
4.8. Изменение записей в таблице.....	115
Задача.....	115
Решение	116
Обсуждение.....	116
4.9. Обновление при условии наличия соответствующих строк.....	117
Задача.....	117
Решение	117
Обсуждение.....	117

4.10. Обновление значениями из другой таблицы.....	118
Задача.....	118
Решение	119
DB2.....	119
MySQL.....	119
Oracle	119
PostgreSQL.....	120
SQL Server	120
Обсуждение.....	120
DB2.....	120
Oracle	121
PostgreSQL, SQL Server и MySQL.....	121
4.11. Слияние записей	122
Задача.....	122
Решение	123
Обсуждение.....	123
4.12. Удаление всех записей таблицы	123
Задача.....	123
Решение	124
Обсуждение.....	124
4.13. Удаление определенных записей	124
Задача.....	124
Решение	124
Обсуждение.....	124
4.14. Удаление одной записи	124
Задача.....	124
Решение	124
Обсуждение.....	125
4.15. Удаление строк, нарушающих ссылочную целостность.....	125
Задача.....	125
Решение	125
Обсуждение.....	125
4.16. Удаление дубликатов записей	126
Задача.....	126
Решение	127
Обсуждение.....	127
4.17. Удаление записей, на которые есть ссылки из другой таблицы	128
Задача.....	128
Решение	129
Обсуждение.....	129
4.18. Подведем итоги	129
ГЛАВА 5. ЗАПРОСЫ НА ПОЛУЧЕНИЕ МЕТАДАННЫХ.....	130
5.1. Создание списка таблиц схемы.....	130
Задача.....	130
Решение	130
DB2.....	131
Oracle	131
PostgreSQL, MySQL и SQL Server.....	131
Обсуждение.....	131

5.2. Создание списка столбцов таблицы.....	131
Задача.....	131
Решение.....	132
DB2.....	132
Oracle.....	132
PostgreSQL, MySQL и SQL Server.....	132
Обсуждение.....	132
5.3. Создание списка индексированных столбцов таблицы.....	132
Задача.....	132
Решение.....	133
DB2.....	133
Oracle.....	133
PostgreSQL.....	133
MySQL.....	133
SQL Server.....	133
Обсуждение.....	134
5.4. Создание списка ограничений таблицы.....	134
Задача.....	134
Решение.....	134
DB2.....	134
Oracle.....	135
PostgreSQL, MySQL и SQL Server.....	135
Обсуждение.....	135
5.5. Создание списка внешних ключей без соответствующих индексов.....	136
Задача.....	136
Решение.....	136
DB2.....	136
Oracle.....	137
PostgreSQL.....	137
MySQL.....	138
SQL Server.....	138
Обсуждение.....	139
5.6. Генерирование кода SQL с помощью средств SQL.....	139
Задача.....	139
Решение.....	139
Обсуждение.....	141
5.7. Описание представлений словаря данных в базе данных Oracle.....	141
Задача.....	141
Решение.....	141
Обсуждение.....	142
5.8. Подведем итоги.....	143
ГЛАВА 6. РАБОТА СО СТРОКАМИ.....	144
6.1. Проход строки.....	144
Задача.....	144
Решение.....	145
Обсуждение.....	145
6.2. Вставка кавычек в строковые константы.....	147
Задача.....	147
Решение.....	147
Обсуждение.....	147

6.3. Подсчет количества вхождений в строку определенного символа	148
Задача.....	148
Решение	148
Обсуждение.....	148
6.4. Удаление символов из строки	149
Задача.....	149
Решение	150
DB2, Oracle, PostgreSQL и SQL Server.....	150
MySQL	150
Обсуждение.....	150
6.5. Разделение цифровых и символьных данных.....	150
Задача.....	150
Решение	151
DB2.....	152
Oracle	152
PostgreSQL.....	152
SQL Server	153
Обсуждение.....	153
6.6. Определение, содержит ли строка только буквенно-цифровые символы.....	155
Задача.....	155
Решение	156
DB2.....	157
MySQL	157
Oracle и PostgreSQL.....	158
SQL Server	158
Обсуждение.....	158
DB2, Oracle, PostgreSQL и SQL Server.....	158
MySQL	159
6.7. Извлечение инициалов из имен	160
Задача.....	160
Решение	160
DB2.....	160
MySQL	160
Oracle и PostgreSQL.....	161
SQL Server	161
Обсуждение.....	161
DB2.....	161
Oracle и PostgreSQL.....	162
MySQL	163
6.8. Сортировка по подстрокам.....	164
Задача.....	164
Решение	165
DB2, Oracle, MySQL и PostgreSQL	165
SQL Server	165
Обсуждение.....	165
6.9. Сортировка по числу в строке	165
Задача.....	165
Решение	166
DB2.....	167
Oracle	167

PostgreSQL.....	167
MySQL.....	167
Обсуждение.....	168
6.10. Создание из строк таблицы списка с разделителями	172
Задача.....	172
Решение	172
DB2.....	173
MySQL.....	173
Oracle	173
PostgreSQL и SQL Server.....	173
Обсуждение.....	174
MySQL.....	174
PostgreSQL и SQL Server.....	174
6.11. Преобразование данных с разделителями в многозначный список оператора <i>/N</i>	175
Задача.....	175
Решение	176
DB2.....	176
MySQL.....	176
Oracle	177
PostgreSQL.....	177
SQL Server	178
Обсуждение.....	178
DB2 и SQL Server.....	178
MySQL.....	179
Oracle	180
PostgreSQL.....	181
6.12. Упорядочение строки по алфавиту	181
Задача.....	181
Решение	182
DB2.....	182
MySQL.....	183
Oracle	183
PostgreSQL.....	183
SQL Server	184
Обсуждение.....	184
SQL Server	184
MySQL.....	185
Oracle	185
PostgreSQL.....	186
6.13. Идентификация числовых подстрок в строке	187
Задача.....	187
Решение	188
DB2.....	188
MySQL.....	189
Oracle	189
PostgreSQL.....	190
SQL Server	190

Обсуждение.....	190
DB2, Oracle и PostgreSQL	191
MySQL	192
6.14. Извлечение <i>n</i>-й подстроки из списка с разделителями	194
Задача.....	194
Решение	195
DB2.....	195
MySQL	195
Oracle	196
PostgreSQL.....	196
SQL Server	196
Обсуждение.....	197
DB2.....	197
MySQL	198
SQL Server	199
Oracle	199
postgresql.....	200
6.15. Парсинг IP-адреса.....	201
Задача.....	201
Решение	201
DB2.....	201
MySQL	202
Oracle	202
PostgreSQL.....	202
SQL Server	203
Обсуждение.....	203
6.16. Сравнение строк по их звучанию	204
Задача.....	204
Решение	205
Обсуждение.....	205
6.17. Обнаружение текста, не совпадающего с шаблоном	205
Задача.....	205
Решение	206
Обсуждение.....	207
6.18. Подведем итоги	208
ГЛАВА 7. ОПЕРАЦИИ С ЧИСЛАМИ	209
7.1. Вычисление среднего	209
Задача.....	209
Решение	209
Обсуждение.....	210
7.2. Определение минимального и/или максимального значения столбца.....	211
Задача.....	211
Решение	211
Обсуждение.....	212
7.3. Суммирование значений столбца.....	213
Задача.....	213
Решение	213
Обсуждение.....	214

7.4. Подсчет строк в таблице	215
Задача.....	215
Решение	215
Обсуждение.....	215
7.5. Подсчет значений столбца	217
Задача.....	217
Решение	217
Обсуждение.....	217
7.6. Вычисление текущей суммы.....	218
Задача.....	218
Решение	218
Обсуждение.....	218
7.7. Вычисление текущего произведения	219
Задача.....	219
Решение	219
Обсуждение.....	220
7.8. Сглаживание последовательности значений	220
Задача.....	220
Решение	221
Обсуждение.....	222
7.9. Вычисление моды.....	222
Задача.....	222
Решение	223
DB2, MySQL, PostgreSQL и SQL Server	223
Oracle	223
Обсуждение.....	223
DB2 и SQL Server.....	223
Oracle	224
7.10. Вычисление медианы.....	225
Задача.....	225
Решение	225
DB2 и PostgreSQL	225
SQL Server	225
MySQL	226
Oracle	226
Обсуждение.....	226
Oracle, PostgreSQL, SQL Server и DB2.....	226
MySQL	227
7.11. Вычисление процентной доли от целого.....	227
Задача.....	227
Решение	227
MySQL и PostgreSQL	227
DB2, Oracle и SQL Server	228
Обсуждение.....	228
MySQL и PostgreSQL	228
DB2, Oracle и SQL Server	228
7.12. Агрегация столбцов, содержащих значения <i>NULL</i>	230
Задача.....	230
Решение	230
Обсуждение.....	230

7.13. Вычисление среднего без учета крайних значений.....	231
Задача.....	231
Решение.....	231
MySQL и PostgreSQL.....	231
DB2, Oracle и SQL Server.....	231
Обсуждение.....	232
MySQL и PostgreSQL.....	232
DB2, Oracle и SQL Server.....	232
7.14. Преобразование буквенно-цифровых строк в числа.....	233
Задача.....	233
Решение.....	233
DB2.....	233
Oracle, SQL Server и PostgreSQL.....	233
MySQL.....	233
Обсуждение.....	234
7.15. Изменение значений в текущей сумме.....	235
Задача.....	235
Решение.....	236
Обсуждение.....	237
7.16. Находим выбросы, используя среднее абсолютное отклонение.....	237
Задача.....	237
Решение.....	238
SQL Server.....	238
PostgreSQL и DB2.....	238
Oracle.....	239
MySQL.....	239
Обсуждение.....	240
7.17. Обнаруживаем аномальные значения, используя закон Бенфорда.....	241
Задача.....	241
Решение.....	242
Обсуждение.....	242
7.18. Подведем итоги.....	243
ГЛАВА 8. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ С ДАТАМИ.....	244
8.1. Сложение и вычитание дней, месяцев и лет.....	244
Задача.....	244
Решение.....	245
DB2.....	245
Oracle.....	245
PostgreSQL.....	245
MySQL.....	246
SQL Server.....	246
Обсуждение.....	246
8.2. Вычисление количества дней между двумя датами.....	247
Задача.....	247
Решение.....	247
DB2.....	247
Oracle и PostgreSQL.....	247
MySQL и SQL Server.....	248
Обсуждение.....	248

8.3. Вычисление количества рабочих дней между двумя датами.....	249
Задача.....	249
Решение.....	249
DB2.....	249
MySQL.....	250
Oracle.....	250
PostgreSQL.....	251
SQL Server.....	251
Обсуждение.....	252
8.4. Вычисление количества месяцев или лет между двумя датами.....	254
Задача.....	254
Решение.....	254
DB2 и MySQL.....	255
Oracle.....	255
PostgreSQL.....	255
SQL Server.....	256
Обсуждение.....	256
DB2, MySQL и PostgreSQL.....	256
Oracle и SQL Server.....	257
8.5. Вычисление количества секунд, минут и часов между двумя датами.....	257
Задача.....	257
Решение.....	257
DB2.....	257
MySQL.....	258
SQL Server.....	258
Oracle и PostgreSQL.....	258
Обсуждение.....	259
8.6. Вычисление повторений каждого дня недели в году.....	259
Задача.....	259
Решение.....	259
DB2.....	259
MySQL.....	260
Oracle.....	261
PostgreSQL.....	261
SQL Server.....	261
Обсуждение.....	262
DB2.....	262
MySQL.....	264
Oracle.....	266
PostgreSQL.....	268
SQL Server.....	270
8.7. Вычисление разницы в днях между датами двух записей.....	272
Задача.....	272
Решение.....	272
DB2.....	272
MySQL и SQL Server.....	272
Oracle.....	273
PostgreSQL.....	273
Обсуждение.....	273
8.8. Подведем итоги.....	277

ГЛАВА 9. РАБОТА С ДАТАМИ	278
9.1. Определение високосного года	278
Задача.....	278
Решение.....	278
DB2.....	279
Oracle.....	279
PostgreSQL.....	279
MySQL.....	280
SQL Server.....	280
Обсуждение.....	280
DB2.....	280
Oracle.....	282
PostgreSQL.....	282
MySQL.....	284
SQL Server.....	285
9.2. Определение количества дней в году	285
Задача.....	285
Решение.....	285
DB2.....	285
Oracle.....	286
PostgreSQL.....	286
MySQL.....	286
SQL Server.....	286
Обсуждение.....	287
DB2.....	287
Oracle.....	287
PostgreSQL.....	287
MySQL.....	288
SQL Server.....	288
9.3. Извлечение из даты единиц времени	288
Задача.....	288
Решение.....	288
DB2.....	289
Oracle.....	289
PostgreSQL.....	289
MySQL.....	290
SQL Server.....	290
Обсуждение.....	290
9.4. Вычисление первого и последнего дней месяца	291
Задача.....	291
Решение.....	291
DB2.....	291
Oracle.....	291
PostgreSQL.....	291
MySQL.....	292
SQL Server.....	292
Обсуждение.....	292
DB2.....	292
Oracle.....	292

PostgreSQL.....	293
MySQL.....	293
SQL Server.....	293
9.5. Вычисление дат определенного дня недели для всего года.....	293
Задача.....	293
Решение.....	293
DB2.....	293
Oracle.....	294
PostgreSQL.....	294
MySQL.....	295
SQL Server.....	295
Обсуждение.....	296
DB2.....	296
Oracle.....	297
PostgreSQL.....	298
MySQL.....	298
SQL Server.....	298
9.6. Вычисление дат первого и последнего вхождения заданного дня недели в месяце.....	299
Задача.....	299
Решение.....	300
DB2.....	300
Oracle.....	300
PostgreSQL.....	301
MySQL.....	301
SQL Server.....	302
Обсуждение.....	302
DB2 и SQL Server.....	302
Oracle.....	304
PostgreSQL и MySQL.....	305
9.7. Создание календаря.....	307
Задача.....	307
Решение.....	307
DB2.....	308
Oracle.....	308
PostgreSQL.....	309
MySQL.....	310
SQL Server.....	311
Обсуждение.....	311
DB2.....	311
Oracle.....	314
MySQL, PostgreSQL и SQL Server.....	317
9.8. Создание списка начальных и конечных дат кварталов года.....	321
Задача.....	321
Решение.....	321
DB2.....	321
Oracle.....	322
PostgreSQL.....	322
MySQL.....	323
SQL Server.....	323

Обсуждение.....	324
DB2.....	324
Oracle	325
PostgreSQL, MySQL и SQL Server.....	325
9.9. Определение начальной и конечной дат для заданного квартала.....	326
Задача.....	326
Решение	326
DB2.....	326
Oracle	327
PostgreSQL.....	327
MySQL	327
SQL Server	328
Обсуждение.....	328
DB2.....	328
Oracle	329
PostgreSQL.....	330
MySQL	331
SQL Server	332
9.10. Дополнение недостающих дат	333
Задача.....	333
Решение	334
DB2.....	334
Oracle	335
PostgreSQL.....	335
MySQL	336
SQL Server	337
Обсуждение.....	337
DB2.....	337
Oracle	338
PostgreSQL.....	339
MySQL	340
SQL Server	340
9.11. Поиск по заданным единицам времени.....	342
Задача.....	342
Решение	342
DB2 и MySQL	342
Oracle и PostgreSQL.....	342
SQL Server	342
Обсуждение.....	343
9.12. Сравнение записей по определенным частям даты.....	343
Задача.....	343
Решение	343
DB2.....	344
Oracle и PostgreSQL.....	344
MySQL	344
SQL Server	344
Обсуждение.....	345
9.13. Выявление наложений диапазонов дат	347
Задача.....	347

Решение	347
DB2, PostgreSQL и Oracle	348
MySQL	348
SQL Server	348
Обсуждение	349
9.14. Подведем итоги	352
ГЛАВА 10. РАБОТА С ДИАПАЗОНАМИ ЗНАЧЕНИЙ	353
10.1. Поиск диапазона последовательных значений	353
Задача	353
Решение	354
Обсуждение	354
10.2. Вычисление разницы между значениями строк одной группы или сегмента	356
Задача	356
Решение	357
Обсуждение	358
10.3. Определение границ диапазона последовательных значений	363
Задача	363
Решение	364
Обсуждение	365
10.4. Вставка пропущенных значений диапазона	366
Задача	366
Решение	366
DB2	367
Oracle	367
PostgreSQL и MySQL	368
SQL Server	368
Обсуждение	369
10.5. Генерирование последовательных числовых значений	370
Задача	370
Решение	371
DB2 и SQL Server	371
Oracle	371
PostgreSQL	372
Обсуждение	372
DB2 и SQL Server	372
Oracle	372
PostgreSQL	373
10.6. Подведем итоги	374
ГЛАВА 11. РАСШИРЕННЫЙ ПОИСК	375
11.1. Разбивка результирующего множества на страницы	375
Задача	375
Решение	375
Обсуждение	376
11.2. Пропускаем <i>n</i> строк таблицы	377
Задача	377
Решение	378
Обсуждение	378

11.3. Использование логики ИЛИ во внешних объединениях	379
Задача.....	379
Решение	380
Обсуждение.....	381
11.4. Определение строк со взаимобратными значениями.....	381
Задача.....	381
Решение	382
Обсуждение.....	382
11.5. Выборка первых <i>n</i> записей с наибольшими значениями	383
Задача.....	383
Решение	383
Обсуждение.....	384
11.6. Выявление строк с наибольшим и наименьшим значениями.....	384
Задача.....	384
Решение	384
Обсуждение.....	385
11.7. Проверка значений из следующих строк	385
Задача.....	385
Решение	386
Обсуждение.....	386
11.8. Смещение значений строк	388
Задача.....	388
Решение	388
Обсуждение.....	388
11.9. Ранжирование результатов.....	390
Задача.....	390
Решение	391
Обсуждение.....	391
11.10. Исключение дубликатов	391
Задача.....	391
Решение	392
Обсуждение.....	392
11.11. Ход конем.....	394
Задача.....	394
Решение	395
DB2 и SQL Server.....	395
Oracle	395
Обсуждение.....	396
DB2 и SQL Server.....	396
Oracle	397
11.12. Создание простых прогнозов.....	400
Задача.....	400
Решение	400
DB2, MySQL и SQL Server.....	400
Oracle	401
PostgreSQL.....	402
MySQL	402
Обсуждение.....	402
DB2, MySQL и SQL Server.....	402

Oracle	404
PostgreSQL.....	407
11.13. Подведем итоги	408
ГЛАВА 12. СОСТАВЛЕНИЕ ОТЧЕТОВ И ФОРМАТИРОВАНИЕ	
РЕЗУЛЬТИРУЮЩИХ МНОЖЕСТВ	409
12.1. Транспонирование результирующего множества в одну строку	409
Задача	409
Решение	410
Обсуждение.....	410
12.2. Транспонирование результирующего множества в несколько строк.....	412
Задача.....	412
Решение	412
Обсуждение.....	413
12.3. Обратное транспонирование результирующего множества	417
Задача.....	417
Решение	418
Обсуждение.....	418
12.4. Обратное транспонирование результирующего множества в один столбец.....	419
Задача.....	419
Решение	420
Обсуждение.....	421
12.5. Исключение повторяющихся значений из результирующего множества.....	423
Задача.....	423
Решение	423
Обсуждение.....	424
12.6. Транспонирование результирующего множества для облегчения вычислений	
 с несколькими строками	425
Задача.....	425
Решение	426
Обсуждение.....	426
12.7. Создание блоков данных фиксированного размера.....	427
Задача.....	427
Решение	427
Обсуждение.....	428
12.8. Создание предопределенного количества блоков данных.....	429
Задача.....	429
Решение	430
Обсуждение.....	430
12.9. Создание горизонтальных гистограмм	430
Задача.....	430
Решение	431
DB2.....	431
Oracle, PostgreSQL и MySQL	431
SQL Server	431
Обсуждение.....	431
12.10. Создание вертикальных гистограмм.....	432
Задача.....	432
Решение	433
Обсуждение.....	433

12.11. Возвращение столбцов, не указанных в операторе <i>GROUP BY</i>	434
Задача.....	434
Решение	435
Обсуждение.....	436
12.12. Вычисление простых подсумм	438
Задача.....	438
Решение	438
DB2 и Oracle.....	438
SQL Server и MySQL	438
PostgreSQL.....	439
Обсуждение.....	439
DB2 и Oracle.....	439
SQL Server и MySQL	440
PostgreSQL.....	441
12.13. Вычисление подсумм для всех возможных сочетаний	441
Задача.....	441
Решение	442
DB2.....	442
Oracle	442
SQL Server.....	443
PostgreSQL.....	443
MySQL.....	444
Обсуждение.....	444
Oracle, DB2 и SQL Server	444
MySQL.....	449
12.14. Выделение строк, не содержащих подсумм	452
Задача.....	452
Решение	453
Обсуждение.....	454
12.15. Маркировка строк с помощью выражений <i>CASE</i>	454
Задача.....	454
Решение	454
Обсуждение.....	455
12.16. Создание разреженной матрицы	456
Задача.....	456
Решение	456
Обсуждение.....	457
12.17. Группирование строк по интервалам времени.....	457
Задача.....	457
Решение	458
Обсуждение.....	459
12.18. Одновременная агрегация разных групп/сегментов	462
Задача.....	462
Решение	462
Обсуждение.....	462
12.19. Агрегирования скользящего диапазона значений	463
Задача.....	463
Решение	464
DB2 и Oracle.....	464

MySQL	465
PostgreSQL и SQL Server	465
Обсуждение	465
DB2, MySQL и Oracle	465
PostgreSQL и SQL Server	468
12.20. Транспонирование результирующего множества, содержащего подсуммы	471
Задача	471
Решение	472
DB2 и Oracle	472
SQL Server	473
PostgreSQL	473
MySQL	474
Обсуждение	474
12.21. Подведем итоги	477
ГЛАВА 13. ИЕРАРХИЧЕСКИЕ ЗАПРОСЫ	478
13.1. Выражение отношений родитель—потомок	479
Задача	479
Решение	479
DB2, Oracle и PostgreSQL	479
MySQL	480
SQL Server	480
Обсуждение	480
13.2. Выражение отношений потомок—родитель—прародитель	483
Задача	483
Решение	483
DB2, PostgreSQL и SQL Server	484
MySQL	484
Oracle	485
Обсуждение	485
DB2, SQL Server, PostgreSQL и MySQL	485
Oracle	486
13.3. Создание иерархического представления таблицы	488
Задача	488
Решение	488
DB2, PostgreSQL и SQL Server	488
MySQL	489
Oracle	489
Обсуждение	490
DB2, MySQL, PostgreSQL и SQL Server	490
Oracle	491
13.4. Выборка всех дочерних строк заданной родительской строки	492
Задача	492
Решение	492
DB2, PostgreSQL и SQL Server	493
Oracle	493
Обсуждение	493
DB2, MySQL, PostgreSQL и SQL Server	493
Oracle	493

13.5. Определение концевых, неконцевых и корневых узлов	494
Задача	494
Решение	494
DB2, PostgreSQL, MySQL и SQL Server	495
Oracle	495
Обсуждение	495
DB2, PostgreSQL, MySQL и SQL Server	495
Oracle	499
13.6. Подведем итоги	502
ГЛАВА 14. НЕСТАНДАРТНЫЕ ПОДХОДЫ	503
14.1. Создание кросс-табличных отчетов с помощью оператора SQL Server <i>PIVOT</i>	503
Задача	503
Решение	503
Обсуждение	504
14.2. Обратное транспонирование кросс-табличных отчетов с помощью оператора SQL Server <i>UNPIVOT</i>	505
Задача	505
Решение	506
14.3. Транспонирование результирующего множества с помощью оператора Oracle <i>MODEL</i>	507
Задача	507
Решение	508
Обсуждение	508
14.4. Извлечение элементов строки, положение которых неизвестно	512
Задача	512
Решение	512
Обсуждение	513
14.5. Вычисление количества дней в году (альтернативное решение для Oracle)	515
Задача	515
Решение	515
Обсуждение	515
14.6. Поиск смешанных буквенно-цифровых строк	516
Задача	516
Решение	516
Обсуждение	517
14.7. Преобразование десятичных чисел в двоичные в Oracle	519
Задача	519
Решение	519
Обсуждение	520
14.8. Транспонирование ранжированного результирующего множества	522
Задача	522
Решение	522
Обсуждение	523
14.9. Добавление заголовка столбца в дважды развернутое результирующее множество	526
Задача	526
Решение	528
Обсуждение	530

14.10. Преобразование скалярного подзапроса в составной подзапрос в Oracle	539
Задача.....	539
Решение.....	539
Обсуждение.....	540
14.11. Парсинг сериализованных данных в строки	541
Задача.....	541
Решение.....	542
Обсуждение.....	543
14.12. Вычисление процентной доли от целого	546
Задача.....	546
Решение.....	547
Обсуждение.....	547
14.13. Проверка на наличие в группе значений определенного значения	548
Задача.....	548
Решение.....	550
Обсуждение.....	550
14.14. Подведем итоги	552

ПРИЛОЖЕНИЕ 1. КРАТКИЙ ОБЗОР ОКОННЫХ ФУНКЦИЙ..... 553

Группировка.....	553
Определение группы в SQL.....	554
Группы не могут быть пустыми.....	555
Группы уникальны.....	555
Значение <i>COUNT</i> никогда не равно нулю.....	558
Парадоксы.....	558
Взаимосвязь между <i>SELECT</i> и <i>GROUP BY</i>	562
Оконные функции.....	565
Простой пример.....	565
Порядок обработки.....	566
Сегменты.....	567
Воздействие значений <i>NULL</i>	569
Когда порядок имеет значение.....	570
Оператор кадрирования.....	572
Текущая сумма <i>RUN_TOTAL2</i>	573
Текущая сумма <i>RUN_TOTAL3</i>	574
Текущая сумма <i>RUN_TOTAL4</i>	574
Завершаем рассмотрение вопроса кадрирования.....	574
Столбец <i>MIN1</i>	575
Столбец <i>MAX1</i>	575
Столбцы <i>MIN2</i> и <i>MAX2</i>	575
Столбцы <i>MIN3</i> и <i>MAX3</i>	576
Столбец <i>MAX4</i>	576
Удобочитаемость + производительность = мощь.....	576
Запросы «в стиле отчетов».....	578

ПРИЛОЖЕНИЕ 2. ПОДЗАПРОСЫ И ОБОБЩЕННЫЕ

ТАБЛИЧНЫЕ ВЫРАЖЕНИЯ.....	581
Подзапросы.....	581
Обобщенные табличные выражения.....	581
Подведем итоги.....	583

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ.....	585
----------------------------------	------------

*Моей маме: ты лучшая!
Спасибо тебе за все*

– Энтони

Клэр, Майе и Леде

– Роберт

Предисловие

SQL — это универсальный язык профессионалов, работающих с данными. В то же самое время он обойден вниманием, которого заслуживает по сравнению с каким-либо другим популярным средством работы с данными. В результате многие разработчики часто пользуются SQL, однако при этом не выходят за рамки простейших запросов. Нередко причиной такого к нему отношения является их заблуждение, что это все, чем SQL располагает.

В этой книге демонстрируются обширные возможности языка SQL, которые могут значительно обогатить инструментарий разработчика. Прочитав ее, вы увидите, как язык SQL можно использовать для статистического анализа, составления отчетов наподобие тех, что создаются с помощью средств бизнес-аналитики, выполнения поиска текстовых данных, осуществления сложного анализа календарных данных и многого другого.

С момента своего выхода первое издание этой книги пользовалось популярностью в качестве «второго учебника по SQL», т. е. учебника, к которому обращаются после овладения основами предмета изучения. У этого издания было много сильных сторон — таких, как обширный круг рассматриваемых вопросов и дружеский стиль изложения.

Но вычислительные технологии развиваются стремительными темпами, даже в случае с такими сформировавшимися средствами, как язык SQL, уходящий корнями в 70-е годы прошлого столетия. И хотя во втором издании книги не рассматриваются какие-либо новые возможности языка, одно из важных его отличий от предыдущего состоит в том, что функции, которые во время первого издания были скорее новшествами и присутствовали лишь в некоторых реализациях языка, теперь стабилизированы и стандартизированы. Это значительно расширило границы для разработки стандартных решений по сравнению с более ранними возможностями.

В этом отношении важно выделить два ключевых аспекта. Во время выхода первого издания книги обобщенные табличные выражения (ОТВ)¹, включая рекурсивные ОТВ, предлагались разве что в паре реализаций языка, однако в настоящее время они доступны во всех его пяти реализациях, рассматриваемых во втором издании. Они были введены для устранения некоторых практических ограничений SQL, ряд которых можно увидеть непосредственно в предлагаемых здесь рецептах. В *при-*

¹ От англ. Common Table Expression, CTE. — *Прим. переводчика.*

ложении 2 к этому изданию, посвященном рекурсивным ОТВ, подчеркивается их важность и объясняется их актуальность.

Кроме этого, во время выхода первого издания оконные функции также были сравнительно новой возможностью и поддерживались не всеми реализациями SQL. Поэтому для их описания потребовалось отдельное приложение, которое перешло и в это издание (см. приложение 1). Однако в настоящее время оконные функции поддерживаются всеми реализациями языка, рассматриваемыми в этой книге. Оконные функции также поддерживаются всеми известными нам реализациями SQL, хотя громадное количество этих реализаций не позволяет гарантировать, что какая-либо из них не откажется поддерживать оконные функции и/или ОТВ.

Кроме стандартизации запросов, всюду, где это было возможно, мы дополнили главы 6 и 7 книги новым материалом. В материале, дополняющем главу 7, рассматриваются новые приложения для анализа данных в рецептах по абсолютному среднему отклонению и закону Бенфорда. А в главу 6 добавлен новый рецепт для поиска данных, соответствующих озвучиваемому тексту. В нее также перенесен материал по регулярным выражениям из главы 14 предыдущего издания.

Для кого предназначена эта книга?

Эта книга может стать полезной любому пользователю SQL, который хочет расширить возможности своих запросов. Тем не менее потенциальные читатели книги должны обладать базовыми знаниями SQL — например, на уровне книги «Изучаем SQL» Алана Бьюли² («Learning SQL», Alan Beaulieu), а в идеале — чтобы решать практические задачи — иметь опыт создания запросов для реальных данных.

За исключением таких общих требований, эта книга будет полезной для всех пользователей SQL, включая разработчиков систем обработки данных, аналитиков данных, специалистов по визуализации данных, бизнес-аналитиков и т. д. Некоторые из этих пользователей могут никогда или очень редко обращаться к базам данных напрямую, но использовать их средства для запроса и извлечения данных в целях их визуализации, бизнес-аналитики или статистического анализа. Книга в основном концентрируется на рассмотрении практических запросов для решения реальных задач. В тех случаях, где некоторое внимание уделяется теории, это делается для прямой поддержки практических составляющих.

Чего нет в этой книге?

Это практическая книга, в основном посвященная тому, как использовать SQL, чтобы понимать данные. Она не охватывает теоретические аспекты баз данных, проектирование баз данных или теорию, лежащую в основе SQL, за исключением тех случаев, когда это требуется для объяснения конкретных примеров или методов.

² См., например, здесь: <https://www.litres.ru/alan-beaulieu/izuchaem-sql-24500342/>. — Прим. редактора.

Также в ней не рассматриваются расширения баз данных для работы с такими типами данных, как XML или JSON. Информацию по этим специализированным темам вы найдете на соответствующих ресурсах.

Платформы и версии SQL

Разработчики постоянно добавляют новые возможности и функциональности в свои продукты. Поэтому необходимо отметить, какие версии различных платформ использовались при подготовке материала этой книги:

- ◆ DB2 11.5;
- ◆ Oracle Database 19 c;
- ◆ PostgreSQL 12;
- ◆ SQL Server 2017;
- ◆ MySQL 8.0.

Таблицы, используемые в книге

В большинстве примеров книги используются две таблицы: EMP (служащие) и DEPT (отдел). Таблица EMP состоит из 14 строк и содержит поля с числовыми, строковыми и календарными данными. А таблица DEPT состоит всего лишь из 4 строк и содержит поля только с числовыми и строковыми данными. Такие таблицы используются во многих учебниках по базам данных и имеют легко понимаемое отношение типа «один-со-многими» между отделами и служащими.

подавляющее большинство решений в этой книге работают с указанными таблицами. Мы никогда не подстраиваем данные для работы с искусственным решением, которое вам маловероятно придется реализовать в действительности, как это делается в некоторых книгах.

Далее показано содержимое таблиц EMP и DEPT соответственно:

```
select * from emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-1980	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-1981	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-1981	1250	500	30
7566	JONES	MANAGER	7839	02-APR-1981	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-1981	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-1981	2850		30
7782	CLARK	MANAGER	7839	09-JUN-1981	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-1982	3000		20
7839	KING	PRESIDENT		17-NOV-1981	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-1981	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-1983	1100		20

7900	JAMES	CLERK	7698	03-DEC-1981	950	30
7902	FORD	ANALYST	7566	03-DEC-1981	3000	20
7934	MILLER	CLERK	7782	23-JAN-1982	1300	10

```
select * from dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Кроме этого, в книге используются четыре сводные таблицы: T1, T10, T100 и T500. Поскольку эти таблицы служат лишь для того, чтобы упростить создание сводок, мы решили не присваивать им значащие имена. Число после буквы «Т» в названии таблицы обозначает количество содержащихся в ней строк. Например, далее приводится содержимое сводных таблиц T1 и T10:

```
select id from t1;
```

ID
1

```
select id from t10;
```

ID
1
2
3
4
5
6
7
8
9
10

Сводные таблицы полезны тем, что они позволяют быстро создавать наборы строк для упрощения запроса.

Кстати, реализации SQL некоторых поставщиков допускают неполные выражения SELECT — например, без предиката FROM. Иногда в этой книге для ясности вместо неполных запросов мы будем использовать вспомогательную таблицу T1 с одной строкой. Это похоже на использование таблицы DUAL СУБД Oracle, но применение таблицы T1 дает нам универсальный способ делать то же самое во всех реализациях SQL, рассматриваемых в этой книге.

Все другие вспомогательные таблицы задействованы в конкретных примерах, и когда они потребуются, мы рассмотрим их в соответствующих главах.

Соглашения, используемые в книге

В этой книге используется ряд соглашений по написанию кода и типографского оформления. Ознакомившись с этими соглашениями, вы сможете лучше понимать содержащийся в ней материал. В этом отношении особенно важны соглашения по написанию кода, поскольку излагать их для каждого рецепта книги будет непрактично. Вместо этого все важные соглашения приводятся здесь.

Типографские соглашения

◆ ВЕРХНИЙ РЕГИСТР

Используется для обозначения в тексте ключевых слов SQL.

◆ нижний регистр

Используется для кода запросов во всех примерах. В других языках программирования, таких как C или Java, нижний регистр задействуется для написания большинства ключевых слов, что нам кажется намного более удобочитаемым, чем верхний регистр. Поэтому код всех запросов будет в нижнем регистре.



Такой значок обозначает подсказку или совет.



Такой значок обозначает общее примечание.



Такой значок обозначает предупреждение или необходимость обратить особое внимание.

Соглашения по написанию кода

Мы предпочитаем всегда записывать код SQL в нижнем регистре — как ключевые слова, так и пользовательские идентификаторы. Например:

```
select empno, ename
  from emp;
```

Но вы можете записывать свой код так, как вам представляется удобным. Например, многие пользователи предпочитают записывать ключевые слова SQL в верхнем регистре. Используйте тот стиль записи кода, который вам больше по душе или который требуется по условиям вашего проекта.

Тем не менее, несмотря на запись кода примеров нижним регистром, в тексте книги ключевые слова и идентификаторы SQL набраны прописными буквами. Это делается с целью выделения этих элементов на фоне обычного текста. Например:

Предыдущий запрос представляет операцию `SELECT` с таблицей `EMP`.

Хотя в этой книге рассматривается работа с базами данных пяти разных поставщиков, мы решили использовать единообразный формат для вывода:

```
EMPNO ENAME
-----
7369 SMITH
7499 ALLEN
...
```

В предикате `FROM` многих решений используются *вложенные запросы*, или *подзапросы*. Согласно требованиям стандарта ANSI SQL, таким запросам необходимо присваивать псевдонимы (не задавать псевдонимы можно только в СУБД Oracle). Поэтому результирующие наборы вложенных запросов в наших решениях обозначаются псевдонимами типа `X` или `Y`:

```
select job, sal
from (select job, max(sal) sal
      from emp
      group by job)x;
```

Буква `X`, следующая за последней закрывающей скобкой, представляет собой имя «таблицы», возвращаемой подзапросом в операторе `FROM`. Использование псевдонимов столбцов — полезное средство для написания самодокументированного кода, но в случае вложенных запросов (для большинства рецептов этой книги) они являются простой формальностью. В качестве псевдонимов обычно принимаются ничего не значащие названия — такие, как `X`, `Y`, `Z`, `TMP1` или `TMP2`. Однако в тех случаях, когда это может способствовать лучшему пониманию примера, используются также и значащие псевдонимы.

Обратите внимание на нумерацию строк кода SQL в разделах «Решение» книги. Например:

```
1 select ename
2    from emp
3   where deptno = 10
```

Номера строк здесь не являются частью синтаксиса кода, а всего лишь служат для идентификации частей запроса в разделах «Обсуждение».

Благодарности за второе издание

В подготовке второго издания этой книги мне помогала команда замечательных людей. Я благодарю Джесс Хаберманн (Jess Haberman), Вирджинию Вильсон (Virginia Wilson), Кейт Галлоуей (Kate Galloway) и Гарри О'Брайяна (Gary O'Brien) из издательства O'Reilly. Отдельное спасибо Николаусу Адамсу (Nicholas Adams) за то, что он постоянно выручал меня в вопросах по Atlas. Большое спасибо нашим техническим редакторам Алану Бьюли (Alan Beaulieu), Скотту Хайнесу (Scott Haines) и Томасу Нильду (Thomas Nield).

Наконец, громадное спасибо членам моей семьи: Клэр, Майе и Лиде (Clare, Maya, Leda) — за их благожелательное отношение к моему полному погружению в работу над этой книгой.

Роберт де Грааф (Robert de Graaf)

Благодарности за первое издание

Издание этой книги было бы невозможным без поддержки, оказанной нам многими людьми. Благодарю свою маму Конни, которой посвящается эта книга. Если бы не твой тяжелый труд и самопожертвование, я бы не стал тем, кто я есть сегодня. Спасибо тебе за все, мама. Я благодарен и признателен тебе за все то, что ты сделала для нас с братом. Мне крупно повезло, что моя мама — это ты.

Спасибо моему брату Джо. Каждый раз, когда я приезжал домой из Балтимора, чтобы взять передышку в работе над книгой, ты всегда был рядом и напоминал мне, как это здорово — не уделять все свое время работе и что мне нужно побыстрее закончить работу над книгой, чтобы возвратиться к более важным в жизни вещам. Ты классный парень, и я уважаю тебя. Я горжусь тобой и почитаю за честь, что ты мой брат.

Спасибо моей замечательной невесте Джорджии. Без твоей поддержки я бы никогда не смог справиться со всеми этими 600 с лишним страницами книги. День за днем ты была рядом, разделяя со мной все хорошее и плохое на этом пути. Я знаю, что для тебя это было так же тяжело, как и для меня. Несмотря на то, что все свое свободное время я уделял работе над книгой, ты относилась к этому с пониманием и поддержкой, за что я вечно тебе благодарен. Спасибо тебе. Я люблю тебя.

Спасибо моим будущим тестю и теще Кики и Джорджу за вашу поддержку на протяжении всего этого нелегкого времени. Всегда, когда я гостил у вас, я чувствовал себя как дома, и вы всегда кормили меня и Джорджию до отвала. Спасибо моим свояченицам, Анне и Кэти. Мне всегда доставляло удовольствие потусоваться с вами, приезжая домой. Это позволяло нам с Джорджией отвлечься от работы над книгой и от Балтимора и отдохнуть, в чем мы так нуждались.

Спасибо моему редактору Джонатану Геннику (Jonathan Gennick), без которого эта книга никогда бы не увидела свет. Джонатан, тебе причитается громадная доля заслуги в выходе этой книги. Ты сделал намного больше, чем обычно требуется от редактора, за что я тебе премного благодарен. Без предложенных тобой рецептов, массы исправлений и юмористического оптимизма, несмотря на надвигающиеся сроки, я бы не смог справиться с этой работой. Я рад тому, что ты был моим редактором, и признателен тебе за предоставленную мне возможность работать с тобой. Было одно удовольствие общаться со специалистом такого высокого технического уровня и с таким обширным опытом, как ты сам — профессиональный администратор баз данных и автор книг о них. Я не думаю, что найдется много редакторов, которые при желании могли бы бросить редактирование и начать работать администратором баз данных (АБД) практически в какой угодно области, как это мог бы сделать ты. Несомненно, наличие специальности АБД в твоём резюме дает тебе

преимущество в области редактирования, т. к. ты хорошо понимаешь, что я хочу сказать даже тогда, когда я сам не могу связно выразить свою мысль. Издательству O'Reilly крупно повезло иметь тебя в числе своих сотрудников, а мне — в качестве редактора.

Я благодарю Алеса Спетика (Ales Spetic) и Джонатана Генника (Jonfthan Gennick) за их книгу «Transact-SQL Cookbook». Исаак Ньютон однажды сказал: «Я мог видеть дальше других лишь потому, что стоял на плечах гигантов». Раздел благодарностей книги «Transact-SQL Cookbook» содержит слова Алеса Спетика, подтверждающие это знаменитое высказывание, которые, я считаю, должна содержать каждая книга по SQL:

Я надеюсь, что эта книга дополнит существующие работы таких выдающихся авторов, как Джо Селко (Joe Celko), Дэвид Розенштейн (David Rozenshtein), Анатолий Абрамович (Anatoly Abramovich), Юджин Бергер (Eugine Berger), Ицик Бен-Ган (Iztik Ben-Gan), Ричард Снодграсс (Richard Snodgrass) и других. В течение многих вечеров я корпел над их трудами, которые являются источником почти всех моих знаний. Когда я пишу эти строки, то осознаю, что на каждый вечер, который я посвятил познанию раскрытых ими секретов, они, должно быть, потратили 10 вечеров, излагая свои знания в единообразной и понимаемой форме. Для меня это большая честь иметь возможность дать что-то взамен обществу SQL.

Спасибо Санжею Мишра (Sanjay Mishra) за его замечательную книгу «Mastering Oracle SQL», а также за то, что он познакомил меня с Джонатаном. Если бы не Санжей, я бы никогда не встретился с Джонатаном и никогда бы не написал эту книгу. Удивительно, как простое сообщение электронной почты может изменить твою жизнь. Благодарю Дэвида Розенштейна — особенно за его книгу «Essence of SQL», из которой я почерпнул твердое понимание того, как надо думать и решать задачи посредством SQL. Также хочу сказать спасибо Дэвиду Розенштейну, Анатолию Абрамовичу и Юджину Бергеру за их книгу «Optimizing Transact-SQL», научившую меня многим продвинутым методам SQL, которыми я пользуюсь в настоящее время.

Кроме этого, благодарю всю команду Wireless Generation — замечательную компанию с замечательными людьми. Также хочу сказать спасибо всем, кто принял участие в обзоре и рецензировании материала книги и поделился советами, чтобы помочь мне завершить ее: Джесси Дейвис (Jesse Davis), Джоэл Паттерсон (Joel Patterson), Филип Зее (Philip Zee), Кэвин Маршалл (Kevin Marshall), Даг Дэниелс (Doug Daniels), Отис Господнетик (Otis Gospodnetic), Кен Ганн (Ken Gunn), Джон Стюарт (John Stewart), Джим Абрамсон (Jim Abramson), Адам Майер (Adam Mayer), Сьюзен Лау (Susan Lau), Алексис Ле-Кок (Alexis Le-Quoc) и Поль Фьюер (Paul Feuer). Спасибо Мэгги Хо (Maggie Ho) за ее внимательную проверку моей работы и чрезвычайно полезные замечания по разделу «Краткий обзор оконных функций». Спасибо Чаку Ван Бурену (Chuck Van Buren) и Джиллиан Гутенбергу (Gillian Gutenberg) за их замечательный совет бегать по утрам. Утренние пробежки помогали мне освежить память и расслабиться. Не думаю, что я смог бы закончить эту книгу, если бы иногда не выделял немного времени на отдых и развлечение.

И в этой связи благодарю Стива Канга (Steve Kang) и Чада Левинсона (Chad Levinson) за терпение, проявленное ими, когда я доставал их своими бесконечными разговорами о разных методах SQL, а они хотели лишь пойти на Юнион-сквер, чтобы после долгого рабочего дня выпить пива и съесть бургер в Heartland Brewery. Также хочу сказать спасибо Аарону Бойду (Aaron Boyd) за его поддержку, ободряющие слова и, самое главное, за хорошие советы. Аарон — искренний, трудолюбивый и очень откровенный парень, такие люди, как он, — ценное приобретение для любой компании. Кроме этого, хочу поблагодарить Оливера Помела (Oliver Pome) за его помощь и поддержку в работе над этой книгой и, в частности, за решение DB2 по созданию списков с разделителями из строк. Оливер смог решить эту задачу, несмотря на отсутствие у него системы DB2, в которой он мог бы проверить его! Я всего лишь объяснил ему принцип работы оператора WITH, и несколько минут спустя у него уже было готовое решение, с которым вы сможете ознакомиться в этой книге.

Джона Харрис (Jonah Harris) и Дэвид Розенштейн также предоставили мне полезные технические рецензии по материалу книги. А Арун Марат (Arun Marathe), Нуно Пинто де Сото (Nuno Pinto de Souto) и Эндрю Одеван (Andrew Odewahn) внесли свой вклад в разработку структуры книги и выбор рецептов для нее на этапе ее планирования. Большое спасибо всем вам за это.

Спасибо Джону Хайду (John Haydu) и команде разработчиков оператора MODEL из корпорации Oracle Corporation за обзор статьи по этому оператору, которую я написал для издательства O'Reilly, что в конечном итоге помогло мне разобраться в тонкостях его работы. Том Кайт (Tom Kyte) из компании Oracle Corporation разрешил мне использовать свою функцию TO_BASE в решении с применением только средств SQL, за что ему большое спасибо. Бруно Денюит (Bruno Denuit) из корпорации Microsoft дал ответы на мои вопросы о функционировании оконных функций, представленных в SQL Server 2005. А Саймон Риггс (Simon Riggs) из PostgreSQL держал меня в курсе новых возможностей SQL в PostgreSQL. Громадное спасибо тебе, Саймон, за это. Зная, что и когда будет выходить, я смог включить в материал книги такие возможности SQL, как изящная функция GENERATE_SERIES, которую я считаю намного более элегантным решением, чем сводные таблицы.

Последнему в этом списке благодарностей, но никак не по значимости, я хочу сказать спасибо Кэю Янгу (Kay Young). Талантливым людям, увлеченным своей работой, всегда по душе работать с другими такими же талантливыми и увлеченными людьми. Многие рецепты в этой книге были созданы в результате совместной работы с Кеем и решения повседневных задач SQL в Wireless Generation. Хочу поблагодарить тебя и сказать, что я глубоко ценю всю помощь, которую ты оказал мне в процессе всего этого. Своими советами, исправлениями грамматических ошибок, участием в разработке кода ты сыграл неоценимую роль в создании этой книги. Было просто замечательно работать с тобой, а компании Wireless Generation здорово повезло иметь в своих рядах такого ценного работника.

Энтони Молинаро (Anthony Molinaro)

Извлечение записей

В этой главе мы научимся работать с самыми простыми операторами `SELECT`. Важно твердо понимать закладываемые здесь основы, поскольку многие рассматриваемые в этой главе задачи не только входят в состав более трудных рецептов книги, но также постоянно возникают в ежедневной работе с SQL.

1.1. Извлечение из таблицы всех строк и столбцов

ЗАДАЧА

Требуется просмотреть все данные, содержащиеся в определенной таблице.

РЕШЕНИЕ

Используйте символ `*` в применяемом к таблице операторе `SELECT`:

```
1 select *
2   from emp
```

Обсуждение

В SQL символ `*` имеет специальное значение — задает возвращение каждого столбца указанной таблицы. Поскольку в приведенном примере не используется предикат `WHERE`, будут возвращены все строки таблицы. Альтернативным подходом было бы явно указать в операторе `SELECT` все столбцы:

```
select empno,ename,job,sal,mgr,hiredate,comm,deptno
   from emp
```

При интерактивной работе с запросами — например, при разработке или отладке программ — более удобно применять подход с использованием оператора `SELECT *`. Но для запросов, исполняющихся в приложениях, лучше указывать каждый столбец по отдельности. При явном указании столбцов мы всегда будем знать, какие именно столбцы возвращаются запросом, но при этом производительность останется такой же. Кроме того, явное указание облегчает понимание запросов другими пользователями (которые могут и не знать названий всех столбцов таблиц в запросе). При использовании оператора `SELECT *` в программном запросе могут возникнуть проблемы, если программа получит от запроса не те столбцы, что ожидалось. По крайней мере, если вы укажете все столбцы, а в возврате один или несколько

столбцов будут отсутствовать, возникшая ошибка с большей вероятностью окажется связана с конкретным отсутствующим столбцом (столбцами).

1.2. Извлечение из таблицы подмножества строк

ЗАДАЧА

Требуется просмотреть только те строки таблицы, которые отвечают определенному условию.

РЕШЕНИЕ

Требуемые строки задаются с использованием предиката `WHERE`. Например, получить список всех служащих отдела 10 можно с помощью следующего запроса:

```
1 select *
2   from emp
3  where deptno = 10
```

Обсуждение

Использование предиката `WHERE` в запросе позволяет возвращать в результате только требуемые строки — т. е. те строки, для которых истинно выражение предиката `WHERE`.

Большинство систем управления базами данных поддерживают обычные условные операторы — такие, как: `=`, `<`, `>`, `<=`, `>=`, `!` и `<>`. Кроме этого, с помощью операторов `AND` (логическое И) и `OR` (логическое ИЛИ) в сочетании с круглыми скобками можно возвращать строки, удовлетворяющие нескольким условиям, как показано в следующем рецепте.

1.3. Возвращение строк по нескольким условиям

ЗАДАЧА

Требуется извлечь из таблицы все строки, отвечающие нескольким условиям.

РЕШЕНИЕ

Используем предикат `WHERE`, объединяя в нем требуемые условия с помощью операторов `OR` и `AND`. Например, вернуть список всех служащих отдела 10 и служащих, получающих комиссию, а также всех служащих отдела 20 с зарплатой не более \$2000 можно с помощью следующего запроса:

```
1 select *
2   from emp
```

```

3 where deptno = 10
4    or comm is not null
5    or sal <= 2000 and deptno=20

```

Обсуждение

Извлечь из таблицы все строки, отвечающие нескольким условиям, можно, объединяя требуемые условия с помощью логических операторов AND и OR и заключая их в круглые скобки. В приведенном примере решения предикат WHERE извлекает строки, в которых:

- ◆ DEPTNO равен 10, или
- ◆ COMM не NULL, или
- ◆ зарплата любого служащего из отдела DEPTNO 20 не превышает \$2000.

При заключении условий в круглые скобки все эти условия проверяются одновременно.

Посмотрим, например, каким будет результирующее множество при заключении условий запроса в круглые скобки:

```

select *
  from emp
 where ( deptno = 10
        or comm is not null
        or sal <= 2000
        )
 and deptno=20

```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-1980	800		20
7876	ADAMS	CLERK	7788	12-JAN-1983	1100		20

1.4. Извлечение из таблицы подмножества столбцов

ЗАДАЧА

Требуется извлечь из таблицы значения не всех, а только определенных столбцов.

РЕШЕНИЕ

Укажите в запросе требуемые столбцы. Например, следующий запрос возвращает только столбцы имени, номера отдела и зарплаты служащего:

```

1 select ename,deptno,sal
2    from emp

```


Обсуждение

Явно указывая в операторе `SELECT` требуемые столбцы, можно обеспечить возвращение только интересующих нас данных. Это особенно важно при обращении к базе данных по сети, поскольку позволяет избежать пустой траты времени, сопутствующей извлечению лишних данных.

1.5. Задание столбцам значимых имен

ЗАДАЧА

Требуется изменить имена столбцов таблицы, возвращаемых запросом, более легко читаемыми и понимаемыми. Рассмотрим следующий запрос, который возвращает зарплаты и комиссионные всех служащих:

```
1 select sal, comm
2   from emp
```

Что означает `sal`? Сокращенное «`sale`» (продажа)? Или же это чье-то имя? А что такое `comm`? Сокращенное «`communication`» (связь)? Очевидно, что имена столбцов результирующего множества должны быть более осмысленными.

РЕШЕНИЕ

Изменить названия столбцов результирующего множества можно с помощью ключевого слова `AS` следующим образом:

```
исходное_имя AS новое_имя
```

Некоторые базы данных не требуют использования `AS`, но при этом все они допускают его:

```
1 select sal as salary, comm as commission
2   from emp
```

```
SALARY  COMMISSION
-----  -
800
1600      300
1250      500
2975
1250      1400
2850
2450
3000
5000
1500      0
1100
 950
3000
1300
```

Обсуждение

Изменение названий столбцов результирующего множества с помощью ключевого слова `AS` известно как *присваивание псевдонимов* (aliasing) этим столбцам. Новые имена столбцов являются псевдонимами.

Качественные псевдонимы значительно улучшают понимание запроса и его результатов пользователями.

1.6. Обращение к столбцу в предикате *WHERE* по его псевдониму

ЗАДАЧА

Из результирующего множества, столбцам которого присвоены псевдонимы, требуется исключить некоторые строки с помощью предиката `WHERE`. Например, следующим образом:

```
select sal as salary, comm as commission
  from emp
 where salary < 5000
```

Но исполнение этого запроса возвращает ошибку.

РЕШЕНИЕ

Эту ошибку можно устранить с помощью вложенного запроса:

```
1 select *
2   from (
3     select sal as salary, comm as commission
4       from emp
5     ) x
6   where salary < 5000
```

Обсуждение

В этом простом примере такой же результат можно получить, не прибегая к использованию вложенного запроса, а обращаясь к столбцам в предикате `WHERE` напрямую по их именам: `COMM` или `SAL`. Но приведенное решение необходимо в тех случаях, когда в предикате `WHERE` надо обращаться к любому из следующих элементов:

- ◆ агрегатные функции;
- ◆ скалярные подзапросы;
- ◆ оконные функции;
- ◆ псевдонимы.

Обращаться к столбцам внешнего запроса по их псевдонимам нам позволяет вложение запроса, присваивающего псевдонимы столбцам, во внешний запрос. Дело в том, что в примере запроса в *разд. «Задача»* предикат `WHERE` обрабатывается перед оператором `SELECT`, когда псевдонимы `SALARY` и `COMMISSION` еще не были созданы, — ведь эти псевдонимы присваиваются столбцам только по завершении обработки предиката `WHERE`. Однако оператор `FROM` обрабатывается перед предикатом `WHERE`. Поэтому при вложении исходного запроса в оператор `FROM` его результаты возвращаются до обработки предиката `WHERE` внешнего запроса, что позволяет этому предикату `WHERE` «видеть» псевдонимы. Такой прием может быть особенно полезным в случае, когда требуется поменять названия столбцов результирующего множества.



В этом решении вложенному запросу присваивается псевдоним `x`. Явного присваивания псевдонима вложенному запросу требуют лишь некоторые базы данных, но все они допускают эту операцию.

1.7. Конкатенация значений столбцов

ЗАДАЧА

Требуется вернуть значения нескольких извлекаемых столбцов в одном столбце результирующего множества. Например, результирующее множество запроса к таблице `EMP` должно выглядеть следующим образом:

```
CLARK WORKS AS A MANAGER
KING WORKS AS A PRESIDENT
MILLER WORKS AS A CLERK
```

Но данные для создания этого результирующего множества берутся из двух разных столбцов таблицы `EMP` — `ENAME` и `JOB`:

```
select ename, job
       from emp
       where deptno = 10
```

ENAME	JOB
-----	-----
CLARK	MANAGER
KING	PRESIDENT
MILLER	CLERK

РЕШЕНИЕ

Для конкатенации значений из нескольких столбцов запроса в один столбец обратимся к соответствующей встроенной функции конкатенации используемой СУБД.

DB2, Oracle, PostgreSQL

В этих СУБД в качестве оператора конкатенации используется двойная вертикальная черта:

```

1 select ename||' WORKS AS A '||job as msg
2   from emp
3  where deptno=10

```

MySQL

Конкатенация в этой базе данных реализуется посредством функции `CONCAT`:

```

1 select concat(ename, ' WORKS AS A ',job) as msg
2   from emp
3  where deptno=10

```

SQL Server

А в этой СУБД используется оператор `+`:

```

1 select ename + ' WORKS AS A ' + job as msg
2   from emp
3  where deptno=10

```

Обсуждение

Для конкатенации значений из нескольких столбцов запроса в один столбец применяется встроенная функция конкатенации используемой СУБД. В СУБД DB2, Oracle и PostgreSQL эта функция сокращенно представляется оператором `||`, в СУБД MySQL — оператором `CONCAT`, а в СУБД SQL Server — оператором `+`.

1.8. Использование условной логики в операторе *SELECT*

ЗАДАЧА

Требуется выполнять операции IF-ELSE со значениями оператора `SELECT`. Например, нужно, чтобы в результирующем множестве для служащих с зарплатой менее \$2000 указывался статус `UNDERPAID` (низкооплачиваемый), с зарплатой \$4000 и больше — `OVERPAID` (высокооплачиваемый), а с зарплатой между этими двумя пределами — `OK`. Результирующее множество должно выглядеть примерно так:

ENAME	SAL	STATUS
SMITH	800	UNDERPAID
ALLEN	1600	UNDERPAID
WARD	1250	UNDERPAID
JONES	2975	OK
MARTIN	1250	UNDERPAID
BLAKE	2850	OK
CLARK	2450	OK
SCOTT	3000	OK
KING	5000	OVERPAID

TURNER	1500 UNDERPAID
ADAMS	1100 UNDERPAID
JAMES	950 UNDERPAID
FORD	3000 OK
MILLER	1300 UNDERPAID

РЕШЕНИЕ

Условную логику в операторе `SELECT` можно реализовать посредством выражения `CASE`:

```
1 select ename,sal,
2     case when sal <= 2000 then 'UNDERPAID'
3         when sal >= 4000 then 'OVERPAID'
4         else 'OK'
5     end as status
6 from emp
```

Обсуждение

Выражение `CASE` позволяет выполнять логические операции с возвращаемыми запросом значениями. Чтобы сделать результирующее множество удобочитаемым, результату выражения `CASE` можно присвоить псевдоним. Так, в приведенном примере результату выражения `CASE` присваивается псевдоним `STATUS`. Оператор `ELSE` использовать не обязательно. При отсутствии оператора `ELSE` выражение `CASE` возвращает значение `NULL` для любой строки, не удовлетворяющей проверяемому условию.

1.9. Ограничение числа возвращаемых строк

ЗАДАЧА

Требуется ограничить количество строк результирующего множества, возвращаемого по вашему запросу. При этом порядок строк не имеет значения: вернуть можно любые n строк.

РЕШЕНИЕ

Для управления количеством возвращаемых строк используется встроенная функция СУБД.

DB2

Для СУБД DB2 это оператор `FETCH FIRST`:

```
1 select *
2 from emp fetch first 5 rows only
```

MySQL и PostgreSQL

Для СУБД MySQL и PostgreSQL то же самое достигается с помощью оператора LIMIT:

```
1 select *
2   from emp limit 5
```

Oracle

В СУБД Oracle количество возвращаемых запросом строк ограничивается использованием функции ROWNUM в предикате WHERE:

```
1 select *
2   from emp
3  where rownum <= 5
```

SQL Server

А в СУБД SQL Server для этого служит ключевое слово TOP:

```
1 select top 5 *
2   from emp
```

Обсуждение

Многие СУБД содержат операторы наподобие FETCH FIRST и LIMIT, позволяющие указывать количество строк, возвращаемых запросом. Но СУБД Oracle отличается использованием для этого функции ROWNUM, которая выдает порядковый номер для каждой возвращаемой строки (возрастающее значение, начинающееся с 1).

При возвращении, например, первых пяти строк функцией ROWNUM <= 5 происходит следующее:

1. Oracle исполняет запрос.
2. Oracle извлекает первую строку и присваивает ей номер 1.
4. Проверяется, равен ли номер строки 5. Если нет, строка отвечает условию, требующему, чтобы номер строки был меньше или равен пяти, и Oracle возвращает строку. Если да, условие не выполняется и строка не возвращается.
5. Oracle извлекает следующую строку и инкрементирует номер строки (до двух, трех, четырех и т. д.).
6. Повторяется шаг 3.

Как можно видеть, значения, возвращаемые функцией ROWNUM, присваиваются после извлечения очередной строки. Это основной ключевой момент происходящего процесса. Часто разработчики на Oracle пытаются вернуть только, например, пятую строку результирующего множества, задавая в запросе ROWNUM = 5.

Но использование условия равенства с функцией ROWNUM не дает желаемого результата. Рассмотрим, что будет, если попытаться вернуть, например, пятую строку, используя выражение ROWNUM = 5:

1. Oracle исполняет запрос.
2. Извлекается первая строка, которой присваивается номер 1.
3. Проверяется, равен ли номер строки 5. Если нет, строка отбрасывается, поскольку она не отвечает заданному условию. Если да, условие выполняется и строка возвращается. Но условие-то выполняться не будет никогда!
4. Oracle извлекает следующую строку, которой опять же присваивается номер 1. Это объясняется тем, что номер первой строки, возвращаемой из запроса, должен быть 1.
5. Повторяется шаг 3.

Внимательно изучив этот процесс, можно понять, почему выражение `ROWNUM = 5` не даст желаемого результата, — чтобы получить строку 5, сначала нужно получить строки с первой по четвертую!

Однако обратите внимание, что выражение `ROWNUM = 1` возвращает первую строку, что может выглядеть противоречащим только что приведенному объяснению работы функции `ROWNUM`. В этом случае требуемый результат возвращается, потому что для определения наличия строк в таблице Oracle должна попытаться извлечь по крайней мере одну строку. Проанализируйте описанный процесс, заменив 5 на 1, и вы сможете понять, почему условие `ROWNUM = 1` дает желаемый результат, возвращая одну строку.

1.10. Извлечение из таблицы произвольных записей

ЗАДАЧА

Требуется вернуть из таблицы определенное количество произвольных записей. Для этого следующий код нужно модифицировать таким образом, чтобы при каждом исполнении он возвращал другое результирующее множество из пяти строк:

```
select ename, job
  from emp
```

РЕШЕНИЕ

Задача решается сортировкой в произвольном порядке строк в операторе `ORDER BY` с привлечением для этого встроенной функции используемой СУБД. А количество возвращаемых строк ограничивается с помощью метода, рассмотренного в *решении 1.9*.

DB2

Используем вместе с `ORDER BY` и `FETCH` встроенную функцию `RAND`:

```
1 select ename, job
2   from emp
3  order by rand() fetch first 5 rows only
```

MySQL

Используем вместе с `LIMIT` и `ORDER BY` встроенную функцию `RAND`:

```
1 select ename, job
2   from emp
3  order by rand() limit 5
```

PostgreSQL

Используем вместе с `LIMIT` и `ORDER BY` встроенную функцию `RANDOM`:

```
1 select ename, job
2   from emp
3  order by random() limit 5
```

Oracle

Используем вместе с `ORDER BY` и встроенной функцией `ROWNUM` встроенную функцию `VALUE`, входящую в состав встроенного пакета `DBMS_RANDOM`:

```
1 select *
2   from (
3    select ename, job
4     from emp
5    order by dbms_random.value()
6           )
7  where rownum <= 5
```

SQL Server

Используем вместе с `TOP` и `ORDER BY` встроенную функцию `NEWID`:

```
1 select top 5 ename, job
2   from emp
3  order by newid()
```

Обсуждение

Порядок строк результирующего множества можно изменить, передавая в оператор `ORDER BY` возвращаемое функцией значение. Во всех приведенных здесь решениях количество возвращаемых строк ограничивается *после* исполнения функции в операторе `ORDER BY`. Пользователям СУБД иных, чем Oracle, может быть полезным изучить решение для этой СУБД, поскольку в нем демонстрируется принцип того, что происходит за кулисами других решений.

Важно видеть разницу в работе оператора `ORDER BY` при использовании в нем функции и числовой константы. А именно: числовая константа задает порядок сортировки по столбцу с соответствующим порядковым номером в списке `SELECT`. А функция задает порядок сортировки по ее результатам, поскольку она вычисляется для каждой строки.

1.11. Поиск значений NULL

ЗАДАЧА

Требуется обнаружить все строки, заданный столбец которых имеет значение NULL (неопределенное значение).

РЕШЕНИЕ

Проверка на NULL выполняется с помощью оператора IS NULL:

```
1 select *
2   from emp
3  where comm is null
```

Обсуждение

Сущность NULL никогда не бывает равной или не равной какой-либо другой сущности, даже самой себе. Поэтому нельзя проверить с помощью оператора = или !=, равно ли содержимое столбца значению NULL. Проверка на наличие в строке значений NULL выполняется с использованием оператора IS NULL. Этот же оператор можно задействовать для выявления строк, не содержащих значение NULL в заданном столбце.

1.12. Преобразование значений NULL в реальные значения

ЗАДАЧА

Вместо содержащихся в строках значений NULL требуется вернуть реальные значения (не-NULL).

РЕШЕНИЕ

Замену значений NULL значениями не-NULL можно выполнить с помощью функции COALESCE:

```
1 select coalesce(comm, 0)
2   from emp
```

Обсуждение

В качестве аргументов функции COALESCE передается одно или несколько значений, а функция возвращает первое значение не-NULL в этом списке. В приведенном решении если значение столбца COMM не равно NULL, то функция возвращает это значение. В противном случае возвращается ноль.

Наилучшим подходом при работе со значениями `NULL` будет задействовать соответствующую встроенную возможность используемой СУБД. Во многих случаях вы обнаружите, что с этой задачей одинаково хорошо справляются несколько функций. Например, можно воспользоваться выражением `CASE`, которое, как и функция `COALESCE`, применимо для всех СУБД:

```
select case
    when comm is not null then comm
    else 0
end
from emp
```

Тем не менее, хотя с помощью выражения `CASE` и можно заменить значения `NULL` реальными значениями, предпочтительней — как следует из приведенных здесь примеров — использовать решение с `COALESCE` благодаря его легкости и краткости.

1.13. Поиск по шаблону

ЗАДАЧА

Требуется извлечь из таблицы строки, соответствующие заданной подстроке или шаблону. Например, рассмотрим следующий запрос и его результирующее множество:

```
select ename, job
    from emp
    where deptno in (10,20)
```

ENAME	JOB
SMITH	CLERK
JONES	MANAGER
CLARK	MANAGER
SCOTT	ANALYST
KING	PRESIDENT
ADAMS	CLERK
FORD	ANALYST
MILLER	CLERK

Нам нужно извлечь из этого результирующего множества только тех служащих, в именах которых есть буква «I» или чье название должности заканчивается на «ER»:

ENAME	JOB
SMITH	CLERK
JONES	MANAGER
CLARK	MANAGER
KING	PRESIDENT
MILLER	CLERK

РЕШЕНИЕ

Используем оператор `LIKE` совместно с SQL-оператором подстановки `%`:

```
1 select ename, job
2   from emp
3  where deptno in (10,20)
4     and (ename like '%I%' or job like '%ER')
```

Обсуждение

При использовании в операции поиска по шаблону `LIKE` оператор `%` возвращает любую последовательность символов. Большинство реализаций SQL также поддерживают оператор `_` (символ подчеркивания), который возвращает любой отдельный символ. Чтобы вернуть любую строку, содержащую символ «I» (в любом месте), нужно заключить шаблон поиска «I» в операторы `%`. Если же шаблон не заключать в операторы `%`, тогда результат запроса будет определяться местом размещения одного оператора `%`. Например, чтобы найти названия должностей, заканчивающихся на «ER», оператор `%` должен предшествовать этому шаблону, а начинающихся с «ER» — следовать после него.

1.14. Подведем итоги

Возможно, приведенные в этой главе рецепты покажутся вам простыми, но они закладывают базис, на котором зиждется все остальное. Извлечение информации является главной целью запросов к базе данных, поэтому практически все, что рассматривается далее в этой книге, основано на этих рецептах.

Сортировка результатов запроса

В этой главе мы рассмотрим, как оформить результаты запроса согласно заданным требованиям. Понимая, как организовать результирующее множество, можно обеспечить получение более удобочитаемых и содержательных данных.

2.1. Возвращение результатов запроса в заданном порядке

ЗАДАЧА

Требуется извлечь имена, должности и зарплаты служащих отдела 10. При этом строки результирующего множества нужно упорядочить по столбцу зарплаты в возрастающем порядке:

ENAME	JOB	SAL
MILLER	CLERK	1300
CLARK	MANAGER	2450
KING	PRESIDENT	5000

РЕШЕНИЕ

Эта задача решается с помощью оператора `ORDER BY`:

```
1 select ename, job, sal
2   from emp
3  where deptno = 10
4  order by sal asc
```

Обсуждение

Оператор `ORDER BY` позволяет упорядочивать строки результирующего множества требуемым способом. В приведенном решении возвращенные запросом строки сортируются по столбцу `SAL` в возрастающем порядке. По умолчанию оператор `ORDER BY` выполняет сортировку в возрастающем порядке, так что здесь оператор `ASC` (ascending, по возрастанию) не является обязательным. Для сортировки в убывающем порядке используется оператор `DESC` (descending, по убыванию):

```
select ename,job,sal
  from emp
 where deptno = 10
 order by sal desc
```

ENAME	JOB	SAL
KING	PRESIDENT	5000
CLARK	MANAGER	2450
MILLER	CLERK	1300

Вместо названия столбца, по которому следует выполнять сортировку, в операторе ORDER BY можно указать номер требуемого столбца в списке оператора SELECT. Столбцы нумеруются слева направо, начиная с 1. Например:

```
select ename,job,sal
  from emp
 where deptno = 10
 order by 3 desc
```

ENAME	JOB	SAL
KING	PRESIDENT	5000
CLARK	MANAGER	2450
MILLER	CLERK	1300

Число 3 в операторе ORDER BY соответствует третьему столбцу списка в операторе SELECT, которым и является столбец SAL.

2.2. Сортировка по нескольким столбцам

ЗАДАЧА

Требуется отсортировать строки результирующего набора сначала по столбцу DEPTNO по возрастанию, а затем по столбцу зарплат SAL по убыванию. Конечный результат должен выглядеть так:

EMPNO	DEPTNO	SAL	ENAME	JOB
7839	10	5000	KING	PRESIDENT
7782	10	2450	CLARK	MANAGER
7934	10	1300	MILLER	CLERK
7788	20	3000	SCOTT	ANALYST
7902	20	3000	FORD	ANALYST
7566	20	2975	JONES	MANAGER
7876	20	1100	ADAMS	CLERK
7369	20	800	SMITH	CLERK
7698	30	2850	BLAKE	MANAGER
7499	30	1600	ALLEN	SALESMAN
7844	30	1500	TURNER	SALESMAN
7521	30	1250	WARD	SALESMAN

7654	30	1250 MARTIN	SALESMAN
7900	30	950 JAMES	CLERK

РЕШЕНИЕ

Задача решается указанием в операторе `ORDER BY` требуемых столбцов сортировки, разделяя их запятыми:

```
1 select empno,deptno,sal,ename,job
2   from emp
3  order by deptno, sal desc
```

Обсуждение

Старшинство столбцов сортировки в списке оператора `ORDER BY` определяется в порядке слева направо. Если столбец сортировки задается по его номеру в списке оператора `SELECT`, тогда число этого номера не должно превышать количества столбцов в списке. Обычно допускается сортировка по столбцу, не указанному в списке оператора `SELECT`, но тогда столбец сортировки в операторе `ORDER BY` нужно указывать явно по имени. Однако при наличии в запросе оператора `GROUP BY` или `DISTINCT` указание столбцов сортировки, отсутствующих в списке оператора `SELECT`, не допускается.

2.3. Сортировка по подстрокам

ЗАДАЧА

Требуется отсортировать результаты запроса по определенным частям строки. Например, извлечь из таблицы `EMP` столбцы имен `ENAME` и должностей `JOB` и упорядочить эти строки по последним двум символам столбца должностей `JOB`. Результирующее множество должно выглядеть следующим образом:

ENAME	JOB
KING	PRESIDENT
SMITH	CLERK
ADAMS	CLERK
JAMES	CLERK
MILLER	CLERK
JONES	MANAGER
CLARK	MANAGER
BLAKE	MANAGER
ALLEN	SALESMAN
MARTIN	SALESMAN
WARD	SALESMAN
TURNER	SALESMAN
SCOTT	ANALYST
FORD	ANALYST

РЕШЕНИЕ

Эта задача решается с помощью соответствующей функции для возвращения подстроки используемой СУБД, размещенной в операторе `ORDER BY`.

DB2, MySQL, Oracle и PostgreSQL

Используем в операторе `ORDER BY` функцию `SUBSTR`:

```
select ename, job
  from emp
 order by substr(job, length(job)-1)
```

SQL Server

Используем в операторе `ORDER BY` функцию `SUBSTRING`:

```
select ename, job
  from emp
 order by substring(job, len(job)-1, 2)
```

Обсуждение

Упорядочить строки результирующего множества по любой части строки можно с помощью функции возвращения подстроки используемой СУБД. Чтобы выполнить сортировку по последним двум символам строки, определяем номер конечного символа строки (который равен длине строки) и вычитаем из него 2. Предпоследний символ строки станет начальной позицией, с которой берутся все символы. Функция `SUBSTRING` СУБД SQL Server отличается от функции `SUBSTR` тем, что для нее нужно указывать количество выбираемых символов, которое передается в третьем параметре. Для приведенного примера подойдет любое число, большее или равное двум.

2.4. Сортировка смешанных буквенно-цифровых данных

ЗАДАЧА

Требуется упорядочить буквенно-цифровые данные таблицы либо по цифровым, либо по буквенным данным. Возьмем, например, следующее представление данных таблицы служащих EMP:

```
create view V
as
select ename||' '||deptno as data
  from emp

select * from V
```

DATA

SMITH 20
ALLEN 30
WARD 30
JONES 20
MARTIN 30
BLAKE 30
CLARK 10
SCOTT 20
KING 10
TURNER 30
ADAMS 20
JAMES 30
FORD 20
MILLER 10

Нам нужно упорядочить эти данные либо по столбцу DEPTNO, либо по столбцу ENAME. Сортировка по столбцу DEPTNO даст следующее результирующее множество:

DATA

CLARK 10
KING 10
MILLER 10
SMITH 20
ADAMS 20
FORD 20
SCOTT 20
JONES 20
ALLEN 30
BLAKE 30
MARTIN 30
JAMES 30
TURNER 30
WARD 30

А результатом сортировки по столбцу ENAME будет следующее множество:

DATA

ADAMS 20
ALLEN 30
BLAKE 30
CLARK 10
FORD 20
JAMES 30
JONES 20
KING 10


```
MARTIN 30
MILLER 10
SCOTT 20
SMITH 20
TURNER 30
WARD 30
```

РЕШЕНИЕ

Oracle, SQL Server и PostgreSQL

Для сортировки строк модифицируем их с помощью функций REPLACE и TRANSLATE:

```
/* СОРТИРОВКА BY DEPTNO */
```

```
1 select data
2   from v
3  order by replace(data,
4                 replace(
5                 translate(data,'0123456789','#####'),'#',''),'')
```

```
/* СОРТИРОВКА ПО ENAME */
```

```
1 select data
2   from v
3  order by replace(
4         translate(data,'0123456789','#####'),'#','')
```

DB2

В DB2 правила неявного преобразования типов более строгие, чем в Oracle или PostgreSQL, поэтому, чтобы обеспечить правильность представления V, необходимо выполнить явное приведение данных столбца DEPTNO к типу CHAR. Вместо того чтобы снова создавать представление V, просто используется вложенный запрос. В следующем решении функции REPLACE и TRANSLATE задействуются так же, как и в решении для Oracle и PostgreSQL, но порядок аргументов для TRANSLATE слегка иной:

```
/* ORDER BY DEPTNO */
```

```
1 select *
2   from (
3  select ename||' '||cast(deptno as char(2)) as data
4   from emp
5        ) v
6  order by replace(data,
7                 replace(
8                 translate(data,'#####','0123456789'),'#',''),'')
```

```

/* ORDER BY ENAME */

1 select *
2   from (
3   select ename||' '||cast(deptno as char(2)) as data
4     from emp
5        ) v
6   order by replace(
7      translate(data,'#####','0123456789'),'#','')

```

MySQL

В настоящее время функция `TRANSLATE` не поддерживается этой платформой, поэтому в ней эта задача решению не подлежит.

Обсуждение

Функции `TRANSLATE` и `REPLACE` удаляют из каждой строки или числовые, или буквенные данные, позволяя без проблем сортировать строки по данным оставшегося типа. Далее на примере решения для СУБД Oracle показаны результаты запроса, который передается оператору `ORDER BY`. Этот подход применим для всех трех используемых СУБД, и только порядок передаваемых DB2 параметров отличает решение для этой СУБД от остальных.

```

select data,
       replace(data,
              replace(
                 translate(data,'0123456789','#####'),'#',''),'') nums,
       replace(
          translate(data,'0123456789','#####'),'#','') chars
from V

```

DATA	NUMS	CHARS
SMITH 20	20	SMITH
ALLEN 30	30	ALLEN
WARD 30	30	WARD
JONES 20	20	JONES
MARTIN 30	30	MARTIN
BLAKE 30	30	BLAKE
CLARK 10	10	CLARK
SCOTT 20	20	SCOTT
KING 10	10	KING
TURNER 30	30	TURNER
ADAMS 20	20	ADAMS
JAMES 30	30	JAMES
FORD 20	20	FORD
MILLER 10	10	MILLER

2.5. Обработка значений *NULL* при сортировке

ЗАДАЧА

Требуется отсортировать результаты запроса к таблице EMP по столбцу COMM, который может содержать значения *NULL*. Вам надо решить, как указывать старшинство при сортировке значений *NULL*. Иными словами, располагать ли их в отсортированном списке последними:

ENAME	SAL	COMM
TURNER	1500	0
ALLEN	1600	300
WARD	1250	500
MARTIN	1250	1400
SMITH	800	
JONES	2975	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
BLAKE	2850	
CLARK	2450	
SCOTT	3000	
KING	5000	

или первыми, перед строками с реальными значениями:

ENAME	SAL	COMM
SMITH	800	
JONES	2975	
CLARK	2450	
BLAKE	2850	
SCOTT	3000	
KING	5000	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
MARTIN	1250	1400
WARD	1250	500
ALLEN	1600	300
TURNER	1500	0

РЕШЕНИЕ

В зависимости от требуемого представления выходных данных и подхода к сортировке значений *NULL* в используемой СУБД, строки со значениями *NULL* можно размещать как в начале отсортированного списка:

```

1 select ename,sal,comm
2   from emp
3  order by 3

```

так и в его конце:

```

1 select ename,sal,comm
2   from emp
3  order by 3 desc

```

Но при таком подходе сортируются также и не-NULL значения столбца сортировки в возрастающем или убывающем порядке, соответственно. Это может или отвечать заданным требованиям, или нет. Например, может потребоваться упорядочить не-NULL значения по иному принципу, чем значения NULL, — например, по убыванию или по возрастанию, со всеми значениями NULL в конце отсортированного таким образом списка. При этом значения столбца можно упорядочить по условию с помощью выражения CASE.

DB2, MySQL, PostgreSQL и SQL Server

Значения NULL можно пометить специальным «флагом» с помощью выражения CASE. При этом такой флаг должен иметь два значения: одно из которых (0) обозначает значения NULL, а другое (1) — значения не-NULL. Затем просто добавить в оператор ORDER BY столбец со значениями флагов. Таким образом можно легко управлять порядком общей сортировки значений NULL относительно значений не-NULL и отдельной сортировкой значений не-NULL.

```

/* ЗНАЧЕНИЯ НЕ-NULL СТОЛБЦА COMM СОРТИРУЮТСЯ ПО ВОЗРАСТАНИЮ,
   ВСЕ ЗНАЧЕНИЯ NULL РАЗМЕЩАЮТСЯ В КОНЦЕ СПИСКА */

```

```

1 select ename,sal,comm
2   from (
3 select ename,sal,comm,
4        case when comm is null then 0 else 1 end as is_null
5   from emp
6        ) x
7  order by is_null desc,comm

```

ENAME	SAL	COMM
TURNER	1500	0
ALLEN	1600	300
WARD	1250	500
MARTIN	1250	1400
SMITH	800	
JONES	2975	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	

```
BLAKE 2850
CLARK 2450
SCOTT 3000
KING 5000
```

```
/* ЗНАЧЕНИЯ НЕ-NULL СТОЛБЦА COMM СОРТИРУЮТСЯ ПО УБЫВАНИЮ,
   ВСЕ ЗНАЧЕНИЯ NULL РАЗМЕЩАЮТСЯ В КОНЦЕ СПИСКА */
```

```
1 select ename,sal,comm
2 from (
3 select ename,sal,comm,
4       case when comm is null then 0 else 1 end as is_null
5 from emp
6 ) x
7 order by is_null desc,comm desc
```

ENAME	SAL	COMM
MARTIN	1250	1400
WARD	1250	500
ALLEN	1600	300
TURNER	1500	0
SMITH	800	
JONES	2975	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
BLAKE	2850	
CLARK	2450	
SCOTT	3000	
KING	5000	

```
/* ЗНАЧЕНИЯ НЕ-NULL СТОЛБЦА COMM СОРТИРУЮТСЯ ПО ВОЗРАСТАНИЮ,
   ВСЕ ЗНАЧЕНИЯ NULL РАЗМЕЩАЮТСЯ В НАЧАЛЕ СПИСКА */
```

```
1 select ename,sal,comm
2 from (
3 select ename,sal,comm,
4       case when comm is null then 0 else 1 end as is_null
5 from emp
6 ) x
7 order by is_null,comm
```

ENAME	SAL	COMM
SMITH	800	
JONES	2975	

CLARK	2450	
BLAKE	2850	
SCOTT	3000	
KING	5000	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
TURNER	1500	0
ALLEN	1600	300
WARD	1250	500
MARTIN	1250	1400

/* ЗНАЧЕНИЯ НЕ-NULL СТОЛБЦА COMM СОРТИРУЮТСЯ ПО УБЫВАНИЮ,
ВСЕ ЗНАЧЕНИЯ NULL РАЗМЕЩАЮТСЯ В НАЧАЛЕ СПИСКА */

```

1 select ename,sal,comm
2   from (
3 select ename,sal,comm,
4        case when comm is null then 0 else 1 end as is_null
5   from emp
6        ) x
7  order by is_null,comm desc

```

ENAME	SAL	COMM
SMITH	800	
JONES	2975	
CLARK	2450	
BLAKE	2850	
SCOTT	3000	
KING	5000	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
MARTIN	1250	1400
WARD	1250	500
ALLEN	1600	300
TURNER	1500	0

Oracle

Предыдущее решение применимо также и для Oracle. Впрочем, возможно и решение, специфичное только для Oracle, основанное на расширениях NULLS FIRST и NULLS LAST оператора ORDER BY, позволяющих располагать значения NULL в начале или в конце списка, независимо от порядка сортировки значений не-NULl:

/* ЗНАЧЕНИЯ НЕ-NULЛ СТОЛБЦА COMM СОРТИРУЮТСЯ ПО ВОЗРАСТАНИЮ,
ВСЕ ЗНАЧЕНИЯ NULL РАЗМЕЩАЮТСЯ В КОНЦЕ СПИСКА */

```
1 select ename,sal,comm
2   from emp
3  order by comm nulls last
```

ENAME	SAL	COMM
TURNER	1500	0
ALLEN	1600	300
WARD	1250	500
MARTIN	1250	1400
SMITH	800	
JONES	2975	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
BLAKE	2850	
CLARK	2450	
SCOTT	3000	
KING	5000	

/* ЗНАЧЕНИЯ НЕ-NULЛ СТОЛБЦА COMM СОРТИРУЮТСЯ ПО ВОЗРАСТАНИЮ,
ВСЕ ЗНАЧЕНИЯ NULL РАЗМЕЩАЮТСЯ В НАЧАЛЕ СПИСКА */

```
1 select ename,sal,comm
2   from emp
3  order by comm nulls first
```

ENAME	SAL	COMM
SMITH	800	
JONES	2975	
CLARK	2450	
BLAKE	2850	
SCOTT	3000	
KING	5000	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
TURNER	1500	0
ALLEN	1600	300
WARD	1250	500
MARTIN	1250	1400

```
/* ЗНАЧЕНИЯ НЕ-NULL СТОЛБЦА COMM СОРТИРУЮТСЯ ПО УБЫВАНИЮ,  
   ВСЕ ЗНАЧЕНИЯ NULL РАЗМЕЩАЮТСЯ В НАЧАЛЕ СПИСКА */
```

```
1 select ename,sal,comm  
2   from emp  
3  order by comm desc nulls first
```

ENAME	SAL	COMM
SMITH	800	
JONES	2975	
CLARK	2450	
BLAKE	2850	
SCOTT	3000	
KING	5000	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
MARTIN	1250	1400
WARD	1250	500
ALLEN	1600	300
TURNER	1500	0

Обсуждение

Если используемая СУБД не располагает средствами для размещения значений NULL столбца в начале или в конце списка без влияния на порядок сортировки значений не-NULL этого столбца (как это возможно в Oracle), для решения такой задачи требуется использовать дополнительный вспомогательный столбец.



На момент подготовки этой книги расширения `NULLS FIRST` и `NULLS LAST` поддерживались в СУБД DB2 для оператора `ORDER BY`, вложенного в оператор `OVER` в оконных функциях, но не для оператора `ORDER BY` для всего результирующего множества.

Целью такого дополнительного (но только в запросе, а не в таблице) столбца является идентификация значений NULL для последующей группировки их вместе в начале или в конце отсортированного списка. Следующий запрос возвращает результирующее множество для запроса, учитывающего значения в таком дополнительном столбце (решение, не специфичное для Oracle):

```
select ename,sal,comm,  
       case when comm is null then 0 else 1 end as is_null  
  from emp
```

ENAME	SAL	COMM	IS_NULL
SMITH	800		0
ALLEN	1600	300	1
WARD	1250	500	1

JONES	2975		0
MARTIN	1250	1400	1
BLAKE	2850		0
CLARK	2450		0
SCOTT	3000		0
KING	5000		0
TURNER	1500	0	1
ADAMS	1100		0
JAMES	950		0
FORD	3000		0
MILLER	1300		0

Используя значения столбца `IS_NULL`, можно легко разместить строки со значением `NULL` столбца `COMM` в начале или в конце списка, не воздействуя на порядок сортировки значений `не-NULL` этого столбца.

2.6. Сортировка по ключу, зависящему от данных

ЗАДАЧА

Требуется отсортировать данные с применением некоего логического условия. Например, если значение `JOB` равно `SALESMAN`, сортировка выполняется по столбцу `COMM`, а в противном случае — по столбцу `SAL`. В итоге результирующее множество должно выглядеть следующим образом:

ENAME	SAL	JOB	COMM
TURNER	1500	SALESMAN	0
ALLEN	1600	SALESMAN	300
WARD	1250	SALESMAN	500
SMITH	800	CLERK	
JAMES	950	CLERK	
ADAMS	1100	CLERK	
MILLER	1300	CLERK	
MARTIN	1250	SALESMAN	1400
CLARK	2450	MANAGER	
BLAKE	2850	MANAGER	
JONES	2975	MANAGER	
SCOTT	3000	ANALYST	
FORD	3000	ANALYST	
KING	5000	PRESIDENT	

РЕШЕНИЕ

Используем в операторе `ORDER BY` выражение `CASE`:

```

1 select ename, sal, job, comm
2   from emp
3  order by case when job = 'SALESMAN' then comm else sal end

```

Обсуждение

Для динамического управления способом сортировки можно использовать выражение `CASE`. Передаваемые оператору `ORDER BY` значения будут иметь следующий вид:

```
select ename, sal, job, comm,
       case when job = 'SALESMAN' then comm else sal end as ordered
from emp
order by 5
```

ENAME	SAL	JOB	COMM	ORDERED
TURNER	1500	SALESMAN	0	0
ALLEN	1600	SALESMAN	300	300
WARD1	250	SALESMAN	500	500
SMITH	800	CLERK		800
JAMES	950	CLERK		950
ADAMS	1100	CLERK		1100
MILLER	1300	CLERK		1300
MARTIN	1250	SALESMAN	1400	1400
CLARK2	450	MANAGER		2450
BLAKE2	850	MANAGER		2850
JONES2	975	MANAGER		2975
SCOTT	3000	ANALYST		3000
FORD	3000	ANALYST		3000
KING	5000	PRESIDENT		5000

2.7. Подведем итоги

Умение организовать сортировку результатов запроса является одним из основных навыков любого пользователя `SQL`. Оператор `ORDER BY` предоставляет мощные возможности сортировки, но, как мы могли видеть в этой главе, для эффективной работы с ним часто требуется выполнить его тонкие настройки. Важно научиться уверенно работать с этим оператором, поскольку он используется во многих рецептах в последующих главах.

Работа с несколькими таблицами

В этой главе мы рассмотрим, как использовать объединения и операции над множествами для комбинирования данных из нескольких таблиц. Объединения являются основой SQL, но операции над множествами не менее важны. Чтобы овладеть навыками работы со сложными запросами, которые рассматриваются далее в этой книге, необходимо сначала научиться работать с объединениями и операциями над множествами.

3.1. Размещение одного набора строк над другим

ЗАДАЧА

Требуется вернуть данные в нескольких таблицах, разместив одно результирующее множество над другим. Наличие общего ключа для таблиц не является обязательным, но их столбцы должны иметь одинаковые типы данных. Например, надо отобразить имена и номера отдела служащих отдела 10 из таблицы EMP и все названия и номера отделов из таблицы DEPT. Результирующее множество должно выглядеть следующим образом:

ENAME_AND_DNAME	DEPTNO
-----	-----
CLARK	10
KING	10
MILLER	10

ACCOUNTING	10
RESEARCH	20
SALES	30
OPERATIONS	40

РЕШЕНИЕ

Объединить строки из нескольких таблиц в одну комбинированную таблицу можно с помощью операции над множествами UNION ALL:

```

1 select ename as ename_and_dname, deptno
2   from emp
3  where deptno = 10
4  union all
5 select '-----', null

```

```

6  from t1
7  union all
8  select dname, deptno
9  from dept

```

Обсуждение

Оператор `UNION ALL` объединяет строки из нескольких таблиц в одно результирующее множество. Как и со всеми операциями над множествами, количество и тип данных элементов в списках операторов `SELECT` должны быть одинаковыми. Например, оба следующих запроса возвратят ошибку:

```

select deptno | select deptno, dname
  from dept   |   from dept
union all     | union all
select ename  | select deptno
  from emp    |   from emp

```

Важно иметь в виду, что в результирующий набор оператора `UNION ALL` попадут все существующие дубликаты строк. Чтобы исключить дубликаты, можно использовать оператор `UNION`. Например, объединение столбцов `EMP.DEPTNO` и `DEPT.DEPTNO` посредством оператора `UNION` возвращает всего лишь четыре строки:

```

select deptno
  from emp
union
select deptno
  from dept

```

```

DEPTNO
-----
10
20
30
40

```

В результате использования оператора `UNION` вместо `UNION ALL` будет, скорее всего, выполнена сортировка, устраняющая дубликаты. Эту особенность следует иметь в виду при работе с большими результирующими множествами. Операция `UNION` примерно эквивалентна следующему запросу, в котором выходные данные операции `UNION ALL` обрабатываются оператором `DISTINCT`:

```

select distinct deptno
  from (
select deptno
  from emp
union all
select deptno
  from dept
  )

```

```

DEPTNO
-----
      10
      20
      30
      40

```

Но оператор `DISTINCT` следует использовать в запросах только в случаях крайней необходимости, так же как и оператор `UNION` вместо оператора `UNION ALL`. Например, хотя в этой книге с целью упрощения представления учебного материала задействуется ограниченное количество таблиц, в реальных условиях запрос к одной таблице лучше реализовывать более подходящим способом.

3.2. Объединение взаимосвязанных строк

ЗАДАЧА

Требуется извлечь строки из нескольких таблиц, объединив их по общему столбцу или по столбцам, содержащим общие значения. Например, надо извлечь имена всех сотрудников отдела 10 вместе с местонахождением отдела каждого сотрудника, но эти данные хранятся в двух разных таблицах. Результирующее множество должно иметь следующий вид:

```

ENAME      LOC
-----
CLARK      NEW YORK
KING       NEW YORK
MILLER     NEW YORK

```

РЕШЕНИЕ

Задача решается объединением таблиц `EMP` и `DEPT` по столбцу `DEPTNO`:

```

1 select e.ename, d.loc
2   from emp e, dept d
3  where e.deptno = d.deptno
4     and e.deptno = 10

```

Обсуждение

Это решение является примером операции *объединения* (`join`), или, более точно, операции *эквιοбъединения* (`equi-join`), которая представляет собой тип операции *внутреннего объединения* (`inner join`). Операция объединения объединяет (как и следовало ожидать от ее названия) строки из двух таблиц в одну. А операция эквιοбъединения объединяет строки на основании условия эквивалентности (равности) — например, где один номер отдела равен другому. Операция внутреннего объединения выполняет настоящее объединение, возвращая в каждой строке данные из соответствующих строк каждой таблицы.

На концептуальном уровне при создании результирующего множества объединения сначала создается декартово произведение (все возможные комбинации строк) таблиц, указанных в операторе FROM:

```
select e.ename, d.loc,
       e.deptno as emp_deptno,
       d.deptno as dept_deptno
from emp e, dept d
where e.deptno = 10
```

ENAME	LOC	EMP_DEPTNO	DEPT_DEPTNO
CLARK	NEW YORK	10	10
KING	NEW YORK	10	10
MILLER	NEW YORK	10	10
CLARK	DALLAS	10	20
KING	DALLAS	10	20
MILLER	DALLAS	10	20
CLARK	CHICAGO	10	30
KING	CHICAGO	10	30
MILLER	CHICAGO	10	30
CLARK	BOSTON	10	40
KING	BOSTON	10	40
MILLER	BOSTON	10	40

Здесь возвращается имя каждого сотрудника отдела 10 из таблицы EMP вместе с каждым номером отдела из таблицы DEPT. Затем конструкция из элементов e.deptno и d.deptno (объединение) в операторе WHERE ограничивает результирующее множество, возвращая только строки с одинаковыми значениями EMP.DEPTNO и DEPT.DEPTNO:

```
select e.ename, d.loc,
       e.deptno as emp_deptno,
       d.deptno as dept_deptno
from emp e, dept d
where e.deptno = d.deptno
and e.deptno = 10
```

ENAME	LOC	EMP_DEPTNO	DEPT_DEPTNO
CLARK	NEW YORK	10	10
KING	NEW YORK	10	10
MILLER	NEW YORK	10	10

Альтернативное решение заключается в явном использовании оператора JOIN (ключевое слово INNER не является обязательным):

```
select e.ename, d.loc
from emp e inner join dept d
on (e.deptno = d.deptno)
where e.deptno = 10
```

Используйте оператор `JOIN` в тех случаях, когда по какой-либо причине предпочтительней вынести логику объединения из оператора `WHERE` в оператор `FROM`. Оба эти подхода отвечают требованиям ANSI и поддерживаются последними версиями всех СУБД, рассматриваемых в этой книге.

3.3. Поиск строк с общими данными в двух таблицах

ЗАДАЧА

Требуется найти и объединить строки с одинаковыми данными в двух таблицах, но объединение надо выполнить по нескольким столбцам. Рассмотрим, например, следующее представление `V`, созданное из таблицы `EMP`:

```
create view V
as
select ename, job, sal
   from emp
  where job = 'CLERK'
```

```
select * from V
```

ENAME	JOB	SAL
SMITH	CLERK	800
ADAMS	CLERK	1100
JAMES	CLERK	950
MILLER	CLERK	1300

Представление `V` содержит только строки с должностью `CLERK`. Но это представление не содержит все возможные столбцы таблицы `EMP`. А нам нужно, чтобы результирующее множество содержало столбцы `EMPNO`, `ENAME`, `JOB`, `SAL` и `DEPTNO` для всех сотрудников в таблице `EMP`, данные которых совпадают с данными в представлении `V`. В частности, результирующее множество должно выглядеть следующим образом:

EMPNO	ENAME	JOB	SAL	DEPTNO
7369	SMITH	CLERK	800	20
7876	ADAMS	CLERK	1100	20
7900	JAMES	CLERK	950	30
7934	MILLER	CLERK	1300	10

РЕШЕНИЕ

Объединяем таблицы по всем столбцам, необходимым для возвращения требуемого результата. В качестве альтернативы, чтобы не выполнять операцию объединения, можно использовать операцию над множествами `INTERSECT`, возвращающую конъюнкцию (общие строки) двух таблиц.

MySQL и SQL Server

Объединяем таблицу EMP и представление V, используя несколько условий объединения:

```
1 select e.empno,e.ename,e.job,e.sal,e.deptno
2   from emp e, V
3  where e.ename = v.ename
4         and e.job = v.job
5         and e.sal = v.sal
```

Альтернативно, объединение с такими же результатами можно получить, используя оператор JOIN:

```
1 select e.empno,e.ename,e.job,e.sal,e.deptno
2   from emp e join V
3     on ( e.ename = v.ename
4         and e.job = v.job
5         and e.sal = v.sal )
```

DB2, Oracle и PostgreSQL

Решение для MySQL и SQL Server также работает и в DB2, Oracle и PostgreSQL. Его следует использовать, если нужно вернуть значения из представления V.

Если возвращать столбцы из представления V нет надобности, тогда можно использовать операцию над множествами INTERSECT с предикатом IN:

```
1 select empno,ename,job,sal,deptno
2   from emp
3  where (ename,job,sal) in (
4     select ename,job,sal from emp
5   intersect
6     select ename,job,sal from V
7  )
```

Обсуждение

Чтобы получить требуемый результат, нужно выбрать для объединения правильные столбцы. Это особенно важно в тех случаях, когда некоторые столбцы в строках таблиц могут содержать общие значения, а другие нет.

Операция над множествами INTERSECT возвращает строки, содержащие общие данные для обоих источников строк. Используя оператор INTERSECT, необходимо сравнивать одинаковое количество элементов одинакового типа данных из обеих таблиц. При работе с операциями над множествами следует иметь в виду, что по умолчанию дубликаты строк не возвращаются.

3.4. Извлечение из одной таблицы значений, отсутствующих в другой

ЗАДАЧА

Требуется найти в одной таблице (назовем ее исходной) значения, которых нет в другой (которую назовем таблицей назначения). Например, в исходной таблице DEPT нужно найти отделы, которых нет в таблице назначения EMP. Так, таблица DEPT содержит отдел 40, которого нет в таблице EMP, следовательно, результирующее множество должно быть следующим:

```
DEPTNO
-----
      40
```

РЕШЕНИЕ

Для решения этой задачи лучше всего задействовать функции, выполняющие операции вычитания множеств. За исключением MySQL, эти операции поддерживаются всеми рассматриваемыми в этой книге СУБД: DB2, PostgreSQL, SQL Server и Oracle. Для MySQL и других СУБД, не поддерживающих функцию вычитания множеств, используется подзапрос, как показано далее.

DB2, PostgreSQL и SQL Server

Используем оператор над множествами EXCEPT:

```
1 select deptno from dept
2 except
3 select deptno from emp
```

Oracle

Используем оператор над множествами MINUS:

```
1 select deptno from dept
2 minus
3 select deptno from emp
```

MySQL

Используем вложенный запрос, который возвращает все номера отделов DEPTNO таблицы EMP внешнему запросу, который ищет в таблице DEPT строки, отсутствующие в результирующем множестве и возвращенные подзапросом:

```
1 select deptno
2   from dept
3  where deptno not in (select deptno from emp)
```

Обсуждение

DB2, PostgreSQL и SQL Server

Встроенные функции вычитания множеств этих СУБД значительно облегчают решение такой задачи. Оператору `EXCEPT` передается первое результирующее множество, из которого он удаляет все строки, присутствующие во втором результирующем множестве. Эта операция во многом похожа на операцию вычитания.

На использование операторов над множествами, включая оператор `EXCEPT`, накладываются определенные ограничения. В частности, типы данных и количество сравниваемых значений в списках обоих операторов `SELECT` должны быть одинаковыми. Кроме этого, оператор `EXCEPT` не возвращает дубликатов и, в отличие от подзапроса с использованием `NOT IN` (см. обсуждение решения для MySQL), может работать со значениями `NULL`. Оператор `EXCEPT` возвращает строки из верхнего запроса (запрос перед `EXCEPT`), которые отсутствуют в нижнем запросе (запрос после `EXCEPT`).

Oracle

Решение для СУБД Oracle идентично решению с использованием оператора `EXCEPT` с единственной разницей, что в Oracle этот оператор вычитания множеств называется `MINUS`. Во всех других отношениях предыдущее объяснение также применимо и к Oracle.

MySQL

Подзапрос возвращает все номера отделов `DEPTNO` из таблицы `EMP`, а внешний запрос возвращает все номера отделов из таблицы `DEPT`, которые не входят (`NOT IN`) в результирующее множество, возвращенное подзапросом.

В решениях MySQL вопросу устранения дубликатов следует уделять должное внимание. В решениях для других платформ дубликаты из результирующего множества удаляются с помощью операторов `EXCEPT` и `MINUS`, обеспечивая возвращение каждого значения `DEPTNO` только один раз. Конечно же, поскольку в приведенном примере столбец `DEPTNO` является ключевым, это только и может быть единственным результатом в любом случае. Но если бы столбец `DEPTNO` не был ключевым, однократное возвращение значений `DEPTNO`, отсутствующих в таблице `EMP`, можно было бы обеспечить с помощью ключевого слова `DISTINCT`:

```
select distinct deptno
  from dept
 where deptno not in (select deptno from emp)
```

При использовании оператора `NOT IN` нужно не упускать из виду значения `NULL`. Возьмем, например, следующую таблицу `NEW_DEPT`:

```
create table new_dept(deptno integer)
insert into new_dept values (10)
insert into new_dept values (50)
insert into new_dept values (null)
```

Запрос, содержащий вложенный запрос с оператором `NOT IN` для нахождения в таблице `DEPT` значений `DEPTNO`, которых нет в таблице `NEW_DEPT`, не возвратит никаких строк:

```
select *
  from dept
 where deptno not in (select deptno from new_dept)
```

Но почему такой запрос не возвращает никаких строк — ведь столбец `DEPTNO` таблицы `NEW_DEPT` не содержит значений 20, 30 и 40? Потому что он содержит значения `NULL`. Подзапрос возвращает три строки со значениями `DEPTNO` 10, 50 и `NULL`. Операторы `IN` и `NOT IN` являются, по сути, операторами `OR` (ИЛИ) и вследствие особенностей обработки значений `NULL` логическими операциями `OR` возвращают разные результаты.

Чтобы разобраться с этим вопросом, возьмем следующие таблицы истинности, где `T` = true (истина), `F` = false (ложь), `N` = null:

OR	T	F	N
T	T	T	T
F	T	F	N
N	T	N	N

NOT	T	F	N
F	T	F	N
T	F	T	N
N	N	N	N

AND	T	F	N
T	T	F	N
F	F	F	F
N	N	F	N

Затем рассмотрим следующий пример использования оператора `IN` и его эквивалент с использованием оператора `OR`:

```
select deptno
  from dept
 where deptno in ( 10,50,null )
```

```
DEPTNO
-----
      10
```

```
select deptno
  from dept
 where (deptno=10 or deptno=50 or deptno=null)
```

```
DEPTNO
```

```
-----
```

```
10
```

Почему они возвращают только значение 10 столбца DEPTNO? Столбец DEPTNO таблицы DEPT содержит четыре значения (10, 20, 30, 40), каждое из которых сравнивается со значениями предиката (deptno=10 or deptno=50 or deptno=null). Согласно ранее приведенным таблицам истинности, для каждого значения DEPTNO (10, 20, 30, 40) предикат дает следующие результаты:

```
DEPTNO=10
(deptno=10 or deptno=50 or deptno=null)
= (10=10 or 10=50 or 10=null)
= (T or F or N)
= (T or N)
= (T)
```

```
DEPTNO=20
(deptno=10 or deptno=50 or deptno=null)
= (20=10 or 20=50 or 20=null)
= (F or F or N)
= (F or N)
= (N)
```

```
DEPTNO=30
(deptno=10 or deptno=50 or deptno=null)
= (30=10 or 30=50 or 30=null)
= (F or F or N)
= (F or N)
= (N)
```

```
DEPTNO=40
(deptno=10 or deptno=50 or deptno=null)
= (40=10 or 40=50 or 40=null)
= (F or F or N)
= (F or N)
= (N)
```

Теперь понятно, почему операторы IN и OR возвращают только значение 10 столбца DEPTNO.

Давайте рассмотрим тот же пример, но с использованием операторов NOT IN и NOT OR:

```
select deptno
   from dept
  where deptno not in ( 10,50,null )
```

```
( нет строк )
```

```
select deptno
       from dept
       where not (deptno=10 or deptno=50 or deptno=null)
```

(нет строк)

Почему в этом случае не возвращается совсем никаких строк? Посмотрим по таблицам истинности, что здесь происходит:

```
DEPTNO=10
NOT (deptno=10 or deptno=50 or deptno=null)
= NOT (10=10 or 10=50 or 10=null)
= NOT (T or F or N)
= NOT (T or N)
= NOT (T)
= (F)
```

```
DEPTNO=20
NOT (deptno=10 or deptno=50 or deptno=null)
= NOT (20=10 or 20=50 or 20=null)
= NOT (F or F or N)
= NOT (F or N)
= NOT (N)
= (N)
```

```
DEPTNO=30
NOT (deptno=10 or deptno=50 or deptno=null)
= NOT (30=10 or 30=50 or 30=null)
= NOT (F or F or N)
= NOT (F or N)
= NOT (N)
= (N)
```

```
DEPTNO=40
NOT (deptno=10 or deptno=50 or deptno=null)
= NOT (40=10 or 40=50 or 40=null)
= NOT (F or F or N)
= NOT (F or N)
= NOT (N)
= (N)
```

В SQL выражение «TRUE или NULL» равно TRUE, но «FALSE или NULL» равно NULL! Вот это обстоятельство и нужно учитывать при работе с предикатами IN и выполнении логических операций OR, когда вычисляемые выражения содержат значения NULL.

Этой проблемы с NOT IN и значениями NULL можно избежать, используя связанный подзапрос в сочетании с предикатом NOT EXISTS. Подзапрос называется «связанным» по той причине, что он обращается к строкам из внешнего запроса. Далее приво-

дится пример альтернативного решения, которое должным образом обрабатывает значения NULL в строках (возвращаясь к исходному запросу в *разд. «Задача»*):

```
select d.deptno
  from dept d
 where not exists (
   select 1
     from emp e
    where d.deptno = e.deptno
 )
```

```
DEPTNO
-----
40
```

```
select d.deptno
  from dept d
 where not exists (
   select 1
     from new_dept nd
    where d.deptno = nd.deptno
 )
```

```
DEPTNO
-----
30
40
20
```

На концептуальном уровне внешний запрос этого решения исследует каждую строку таблицы DEPT. Для значений DEPT каждой строки выполняются следующие операции:

1. Исполняется подзапрос, проверяющий наличие того или иного номера отдела в таблице EMP. Обратите внимание на условие `D.DEPTNO = E.DEPTNO`, которое сводит вместе номера отделов из обеих таблиц.
2. Если подзапрос возвращает результаты, выражение `EXISTS (...)` возвращает TRUE, а выражение `NOT EXISTS (...)` — соответственно — FALSE, и обрабатываемая внешним запросом строка отбрасывается.
3. Если подзапрос не возвращает результатов, выражение `NOT EXISTS (...)` возвращает TRUE, и обрабатываемая внешним запросом строка возвращается (т. к. значение ее столбца DEPT отсутствует в таблице EMP).

При использовании связанного подзапроса с выражениями `EXISTS/NOT EXISTS` элементы списка `SELECT` подзапроса не имеют значения. Поэтому мы решили выбрать NULL, чтобы заставить вас сфокусироваться на объединении в подзапросе, а не на элементах списка оператора `SELECT`.

3.5. Извлечение строк из таблицы, не соответствующих строкам в другой таблице

ЗАДАЧА

Из одной из двух таблиц с общим ключом нужно извлечь строки, в которых нет совпадающих данных в другой таблице. Например, нам нужно узнать, в каких отделах нет служащих. Результирующее множество должно выглядеть следующим образом:

DEPTNO	DNAME	LOC
40	OPERATIONS	BOSTON

Получить отдел для каждого служащего можно, выполнив эквиобъединение по столбцу `DEPTNO` таблицы `EMP` с таблицей `DEPT`. Столбец `DEPTNO` содержит значения, общие для обеих таблиц. К сожалению, операция эквиобъединения таблиц `EMP` и `DEPT` не покажет отделов, не имеющих служащих, поскольку она возвратит все строки, удовлетворяющие условию объединения. А нам нужны только те строки из таблицы `DEPT`, которые не удовлетворяют этому условию.

Хотя с первого взгляда эта задача может показаться такой же, как и предыдущая, в действительности она слегка отличается от нее. Это отличие состоит в том, что предыдущий рецепт создает список только номеров отделов, которых нет в таблице `EMP`. Этот же рецепт позволит нам возвращать не только номера отделов, но и другие столбцы из таблицы `DEPT`.

РЕШЕНИЕ

Сначала возвращаем все строки из обеих таблиц, независимо от того, есть ли в общем столбце второй таблицы значения, совпадающие со значениями в этом столбце первой таблицы. Затем из полученного результирующего множества оставляем только строки, у которых не совпадают значения общего столбца.

DB2, MySQL, PostgreSQL и SQL Server

Используем внешнее объединение и отфильтровываем значения `NULL` (ключевое слово `OUTER` не обязательно):

```
1 select d.*
2   from dept d left outer join emp e
3     on (d.deptno = e.deptno)
4  where e.deptno is null
```

Обсуждение

Это решение выполняет операцию внешнего объединения, из результатов которого отфильтровываются требуемые строки с разными значениями общего столбца. Такие операции иногда называются *антиобъединением* (*anti-join*). Чтобы разобраться

с принципом работы операции антиобъединения, сначала рассмотрим результирующее множество, из которого еще не были отфильтрованы значения NULL:

```
select e.ename, e.deptno as emp_deptno, d.*
  from dept d left join emp e
    on (d.deptno = e.deptno)
```

ENAME	EMP_DEPTNO	DEPTNO	DNAME	LOC
SMITH	20	20	RESEARCH	DALLAS
ALLEN	30	30	SALES	CHICAGO
WARD	30	30	SALES	CHICAGO
JONES	20	20	RESEARCH	DALLAS
MARTIN	30	30	SALES	CHICAGO
BLAKE	30	30	SALES	CHICAGO
CLARK	10	10	ACCOUNTING	NEW YORK
SCOTT	20	20	RESEARCH	DALLAS
KING	10	10	ACCOUNTING	NEW YORK
TURNER	30	30	SALES	CHICAGO
ADAMS	20	20	RESEARCH	DALLAS
JAMES	30	30	SALES	CHICAGO
FORD	20	20	RESEARCH	DALLAS
MILLER	10	10	ACCOUNTING	NEW YORK
		40	OPERATIONS	BOSTON

Обратите внимание на значения NULL столбцов ENAME и EMP_DEPTNO в последней строке. Наличие этих значений объясняется тем, что в отделе 40 никто не работает. Эта строка результирующего множества, в которой столбец EMP_DEPTNO имеет значение NULL, и выбирается она посредством предиката WHERE (возвращая, таким образом, только те строки таблицы DEPT, в которых значение столбца DEPTNO не совпадает со значением этого столбца таблицы EMP).

3.6. Добавление в запрос независимых объединений

ЗАДАЧА

У вас есть запрос, который возвращает желаемые результаты. Требуется модифицировать его, чтобы он возвращал дополнительную информацию, но в результате попытки выполнить такую модификацию теряются данные из первоначального результирующего множества. Например, нужно получить имена всех служащих, местонахождение их отделов и дату получения премии. Данные о премиях для этой задачи содержатся в таблице EMP_BONUS:

```
select * from emp_bonus
```

EMPNO	RECEIVED	TYPE
7369	14-MAR-2005	1
7900	14-MAR-2005	2
7788	14-MAR-2005	3

Первоначальный запрос выглядит следующим образом:

```
select e.ename, d.loc
       from emp e, dept d
       where e.deptno=d.deptno
```

ENAME	LOC
SMITH	DALLAS
ALLEN	CHICAGO
WARD	CHICAGO
JONES	DALLAS
MARTIN	CHICAGO
BLAKE	CHICAGO
CLARK	NEW YORK
SCOTT	DALLAS
KING	NEW YORK
TURNER	CHICAGO
ADAMS	DALLAS
JAMES	CHICAGO
FORD	DALLAS
MILLER	NEW YORK

Нам нужно модифицировать этот запрос, чтобы он также возвращал и дату выдачи премии служащим. Но объединение с таблицей EMP_BONUS возвращает не все строки, поскольку премию получили не все служащие:

```
select e.ename, d.loc,ab.received
       from emp e, dept d, emp_bonus ab
       where e.deptno=d.deptno
              and e.empno=ab.empno
```

ENAME	LOC	RECEIVED
SCOTT	DALLAS	14-MAR-2005
SMITH	DALLAS	14-MAR-2005
JAMES	CHICAGO	14-MAR-2005

Нам же требуется получить следующее результирующее множество:

ENAME	LOC	RECEIVED
ALLEN	CHICAGO	
WARD	CHICAGO	
MARTIN	CHICAGO	
JAMES	CHICAGO	14-MAR-2005
TURNER	CHICAGO	
BLAKE	CHICAGO	
SMITH	DALLAS	14-MAR-2005
FORD	DALLAS	
ADAMS	DALLAS	
JONES	DALLAS	

SCOTT	DALLAS	14-MAR-2005
CLARK	NEW YORK	
KING	NEW YORK	
MILLER	NEW YORK	

РЕШЕНИЕ

Возвратить дополнительные данные, не теряя при этом данные исходного запроса, можно посредством внешнего объединения. Для этого сначала объединяем таблицу EMP с таблицей DEPT, чтобы получить список всех служащих и местонахождения их отделов. Затем выполняем внешнее объединение этой объединенной таблицы с таблицей EMP_BONUS, чтобы вернуть дату бонуса для тех служащих, которые их получили. В СУБД DB2, MySQL, PostgreSQL и SQL эти операции выполняются посредством следующего запроса:

```
1 select e.ename, d.loc, eb.received
2   from emp e join dept d
3     on (e.deptno=d.deptno)
4  left join emp_bonus eb
5     on (e.empno=eb.empno)
6  order by 2
```

Внешнее объединение можно также эмулировать посредством скалярного подзапроса (подзапроса в списке оператора SELECT):

```
1 select e.ename, d.loc,
2       (select eb.received from emp_bonus eb
3        where eb.empno=e.empno) as received
4   from emp e, dept d
5  where e.deptno=d.deptno
6  order by 2
```

Это решение поддерживается всеми платформами.

Обсуждение

Внешнее объединение возвращает все строки одной таблицы и строки с совпадающими данными заданного столбца другой таблицы. Другой пример такого объединения рассматривался в предыдущем рецепте. Внешнее объединение позволяет решить эту задачу, поскольку оно не вызывает удаления никаких строк, и, кроме строк с датой получения премии, запрос также возвращает все строки, которые он бы возвратил и без него.

Для решения задач такого типа также хорошо подходит использование скалярного подзапроса, поскольку это не требует модифицирования уже работающих должным образом объединений в главном запросе. С помощью такого подзапроса можно легко добавить в существующий запрос новые данные, не нарушая этим целостности текущего результирующего множества. Но скалярные подзапросы должны возвращать только скалярное (одно) значение. Возвращение подзапросом в списке SELECT более чем одной строки вызовет ошибку.

См. также

В рецепте 14.10 рассматривается подход, позволяющий возвращать несколько строк подзапросом в списке оператора `SELECT`.

3.7. Проверка двух таблиц на идентичность

ЗАДАЧА

Требуется проверить две таблицы или представления на идентичность по количеству и значению строк. Возьмем следующее представление:

```
create view V
as
select * from emp where deptno != 10
union all
select * from emp where ename = 'WARD'

select * from V
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-2005	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-2006	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30
7566	JONES	MANAGER	7839	02-APR-2006	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-2006	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-2006	2850		30
7788	SCOTT	ANALYST	7566	09-DEC-2007	3000		20
7844	TURNER	SALESMAN	7698	08-SEP-2006	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-2008	1100		20
7900	JAMES	CLERK	7698	03-DEC-2006	950		30
7902	FORD	ANALYST	7566	03-DEC-2006	3000		20
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30

Нам нужно определить, содержит ли это представление точно такие же данные, как и таблица `EMP`. Мы намеренно продублировали строку для служащего `WARD`, чтобы продемонстрировать, что решение выявит не только разные данные, но также и дубликаты строк. Это представление, таким образом, отличается от таблицы `EMP` отсутствием в нем трех записей для служащих из отдела 10 и дубликатом записи для служащего `WARD`. Результирующее множество при этом должно иметь следующий вид:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30	1
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30	2
7782	CLARK	MANAGER	7839	09-JUN-2006	2450		10	1
7839	KING	PRESIDENT		17-NOV-2006	5000		10	1
7934	MILLER	CLERK	7782	23-JAN-2007	1300		10	1

РЕШЕНИЕ

В зависимости от используемой СУБД, проблема сравнения таблиц достаточно легко решается с помощью функций вычитания множеств `MINUS` или `EXCEPT`. В случае СУБД, не поддерживающих эти функции, можно использовать связанный подзапрос.

DB2 и PostgreSQL

Используя оператор вычитания множеств `EXCEPT`, находим различия между представлением `V` и таблицей `EMP` и между таблицей `EMP` и представлением `V` и объединяем полученные результирующие множества с помощью оператора `UNION ALL`:

```

1 (
2  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
3         count(*) as cnt
4  from V
5  group by empno,ename,job,mgr,hiredate,sal,comm,deptno
6  except
7  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
8         count(*) as cnt
9  from emp
10 group by empno,ename,job,mgr,hiredate,sal,comm,deptno
11 )
12 union all
13 (
14  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
15         count(*) as cnt
16  from emp
17  group by empno,ename,job,mgr,hiredate,sal,comm,deptno
18  except
19  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
20         count(*) as cnt
21  from v
22  group by empno,ename,job,mgr,hiredate,sal,comm,deptno
23 )

```

Oracle

Используя оператор вычитания множеств `MINUS`, находим различия между представлением `V` и таблицей `EMP` и между таблицей `EMP` и представлением `V` и объединяем полученные результирующие множества с помощью оператора `UNION ALL`:

```

1 (
2  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
3         count(*) as cnt
4  from V
5  group by empno,ename,job,mgr,hiredate,sal,comm,deptno

```

```

6  minus
7  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
8         count(*) as cnt
9  from emp
10 group by empno,ename,job,mgr,hiredate,sal,comm,deptno
11 )
12 union all
13 (
14 select empno,ename,job,mgr,hiredate,sal,comm,deptno,
15        count(*) as cnt
16  from emp
17 group by empno,ename,job,mgr,hiredate,sal,comm,deptno
18 minus
19 select empno,ename,job,mgr,hiredate,sal,comm,deptno,
20        count(*) as cnt
21  from v
22 group by empno,ename,job,mgr,hiredate,sal,comm,deptno
23 )

```

MySQL и SQL Server

Для этих СУБД строки в представлении V, отсутствующие в таблице EMP, и наоборот, находим с помощью связанного подзапроса, а полученные результирующие множества так же объединяем с помощью оператора UNION ALL:

```

1  select *
2  from (
3  select e.empno,e.ename,e.job,e.mgr,e.hiredate,
4         e.sal,e.comm,e.deptno, count(*) as cnt
5  from emp e
6  group by empno,ename,job,mgr,hiredate,
7         sal,comm,deptno
8  ) e
9  where not exists (
10 select null
11  from (
12 select v.empno,v.ename,v.job,v.mgr,v.hiredate,
13        v.sal,v.comm,v.deptno, count(*) as cnt
14  from v
15 group by empno,ename,job,mgr,hiredate,
16        sal,comm,deptno
17  ) v
18 where v.empno = e.empno
19  and v.ename = e.ename
20  and v.job = e.job
21  and coalesce(v.mgr,0) = coalesce(e.mgr,0)
22  and v.hiredate = e.hiredate
23  and v.sal = e.sal

```

```
24     and v.deptno = e.deptno
25     and v.cnt = e.cnt
26     and coalesce(v.comm,0) = coalesce(e.comm,0)
27 )
28 union all
29 select *
30 from (
31 select v.empno,v.ename,v.job,v.mgr,v.hiredate,
32        v.sal,v.comm,v.deptno, count(*) as cnt
33 from v
34 group by empno,ename,job,mgr,hiredate,
35          sal,comm,deptno
36        ) v
37 where not exists (
38 select null
39 from (
40 select e.empno,e.ename,e.job,e.mgr,e.hiredate,
41        e.sal,e.comm,e.deptno, count(*) as cnt
42 from emp e
43 group by empno,ename,job,mgr,hiredate,
44          sal,comm,deptno
45        ) e
46 where v.empno = e.empno
47     and v.ename = e.ename
48     and v.job = e.job
49     and coalesce(v.mgr,0) = coalesce(e.mgr,0)
50     and v.hiredate = e.hiredate
51     and v.sal = e.sal
52     and v.deptno = e.deptno
53     and v.cnt = e.cnt
54     and coalesce(v.comm,0) = coalesce(e.comm,0)
55 )
```

Обсуждение

Несмотря на разные методы решений, все они основаны на одинаковой логике:

1. Находим строки в таблице EMP, которых нет в представлении V.
2. Объединяем (UNION ALL) эти строки со строками в представлении V, которых нет в таблице EMP.

Если сравниваемые объекты идентичны, запрос не возвращает никаких строк. Если таблицы различаются, то возвращаются строки, создающие различие. Сравнение таблиц проще всего начать, сравнивая только количество строк в них, не усложняя задачу одновременным сравнением данных.

Следующий простой пример соответствующего запроса может выполняться на всех СУБД:

```
select count(*)
  from emp
 union
select count(*)
  from dept
```

```
COUNT(*)
-----
      4
     14
```

Поскольку оператор `UNION` удаляет дубликаты, для таблиц с одинаковой кардинальностью (мощностью) возвращается только одна строка. В приведенном примере возвращаются две строки, а это означает, что таблицы не содержат идентичных наборов строк.

DB2, Oracle и PostgreSQL

Функции `MINUS` и `EXCEPT` работают идентично, поэтому для обсуждения воспользуемся только функцией `EXCEPT`. Запросы до и после функции `EXCEPT` похожи друг на друга, поэтому, чтобы разобраться с работой решения, рассмотрим работу только запроса, предшествующего функции `EXCEPT`, — это строки 1–11 решений для указанных СУБД:

```
(
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
         count(*) as cnt
  from V
 group by empno,ename,job,mgr,hiredate,sal,comm,deptno
 except
 select empno,ename,job,mgr,hiredate,sal,comm,deptno,
         count(*) as cnt
  from emp
 group by empno,ename,job,mgr,hiredate,sal,comm,deptno
)
```

И исполнение их дает следующее результирующее множество:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30	2

Это результирующее множество выводит строку в представлении `V`, которая или отсутствует в таблице `EMP` или имеет иную кардинальность, чем соответствующая строка в таблице `EMP`. В нашем случае запрос находит и возвращает дубликат строки для служащего `WARD`. Если вы все еще не можете понять, каким образом создается результирующее множество, выполните по отдельности каждый запрос по обе стороны оператора `EXCEPT`. Вы увидите, что единственной разницей между результирующими множествами этих запросов является значение `CNT` для служащего `WARD`, возвращаемого из представления `V`.

Часть запроса после оператора UNION ALL выполняет действия, противоположные действию части запроса до этого оператора: находит строки в таблице EMP, которых нет в представлении V:

```
(
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
         count(*) as cnt
     from emp
  group by empno,ename,job,mgr,hiredate,sal,comm,deptno
 minus
select empno,ename,job,mgr,hiredate,sal,comm,deptno,
         count(*) as cnt
     from v
  group by empno,ename,job,mgr,hiredate,sal,comm,deptno
)
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30	1
7782	CLARK	MANAGER	7839	09-JUN-2006	2450		10	1
7839	KING	PRESIDENT		17-NOV-2006	5000		10	1
7934	MILLER	CLERK	7782	23-JAN-2007	1300		10	1

Результаты этих двух запросов объединяются оператором UNION ALL, возвращая конечное результирующее множество.

MySQL и SQL Server

Запросы до и после оператора UNION ALL похожи друг на друга, поэтому, чтобы разобраться с работой решения, рассмотрим работу только запроса, предшествующего оператору UNION ALL, — это строки 1–27 решения для этих СУБД:

```
select *
  from (
select e.empno,e.ename,e.job,e.mgr,e.hiredate,
       e.sal,e.comm,e.deptno, count(*) as cnt
     from emp e
  group by empno,ename,job,mgr,hiredate,
           sal,comm,deptno
       ) e
 where not exists (
select null
  from (
select v.empno,v.ename,v.job,v.mgr,v.hiredate,
       v.sal,v.comm,v.deptno, count(*) as cnt
     from v
  group by empno,ename,job,mgr,hiredate,
           sal,comm,deptno
       ) v
```



```

where v.empno = e.empno
   and v.ename = e.ename
   and v.job = e.job
   and v.mgr = e.mgr
   and v.hiredate = e.hiredate
   and v.sal = e.sal
   and v.deptno = e.deptno
   and v.cnt = e.cnt
   and coalesce(v.comm,0) = coalesce(e.comm,0)
)

```

И исполнение их дает следующее результирующее множество:

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30	1
7782	CLARK	MANAGER	7839	09-JUN-2006	2450		10	1
7839	KING	PRESIDENT		17-NOV-2006	5000		10	1
7934	MILLER	CLERK	7782	23-JAN-2007	1300		10	1

Обратите внимание, что сравниваются не таблица EMP и представление V, а вложенные представления E и V. Количество элементов для каждой строки определяется и возвращается в виде атрибута этой строки. Мы сравниваем каждую строку и количество ее вхождений. Если вы все еще не можете понять механизм этого сравнения, выполните каждый подзапрос по отдельности. На следующем этапе найдутся все строки (включая CNT) во вложенном представлении E, которые отсутствуют во вложенном представлении V. Это сравнение выполняется посредством связанного подзапроса и операции NOT EXISTS. Объединения выявляют одинаковые строки, а результатом являются все строки во вложенном представлении E, которые отсутствуют в наборе строк, возвращенном объединением. Часть запроса после оператора UNION ALL выполняет действия, противоположные действиям части запроса до этого оператора: находит все строки во вложенном представлении V, которых нет во вложенном представлении E:

```

select *
  from (
select v.empno,v.ename,v.job,v.mgr,v.hiredate,
       v.sal,v.comm,v.deptno, count(*) as cnt
  from v
 group by empno,ename,job,mgr,hiredate,
          sal,comm,deptno
        ) v
 where not exists (
select null
  from (
select e.empno,e.ename,e.job,e.mgr,e.hiredate,
       e.sal,e.comm,e.deptno, count(*) as cnt
  from emp e
 group by empno,ename,job,mgr,hiredate,
          sal,comm,deptno
        ) e

```

```

where v.empno = e.empno
   and v.ename = e.ename
   and v.job = e.job
   and v.mgr = e.mgr
   and v.hiredate = e.hiredate
   and v.sal = e.sal
   and v.deptno = e.deptno
   and v.cnt = e.cnt
   and coalesce(v.comm,0) = coalesce(e.comm,0)
)

```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
7521	WARD	SALESMAN	7698	22-FEB-2006	1250	500	30	2

Результаты этих двух подзапросов объединяются оператором UNION ALL, возвращая конечное результирующее множество.



Альтернативное решение этой задачи приводится в разд. «*Comparing Two Sets for Equality*» («Проверка двух множеств на равенство») главы 2 книги «*Transact-SQL Cookbook*» (авторы Ales Spetic и Jonathan Gennick, издательство O'Reilly).

3.8. Выявление и устранение проблемы декартовых произведений

ЗАДАЧА

Требуется извлечь имена всех сотрудников отдела 10 вместе с местонахождением отдела каждого сотрудника. Можно было бы попробовать решить эту задачу посредством следующего запроса:

```

select e.ename, d.loc
   from emp e, dept d
  where e.deptno = 10

```

Но такой запрос возвращает неправильные данные:

ENAME	LOC
CLARK	NEW YORK
CLARK	DALLAS
CLARK	CHICAGO
CLARK	BOSTON
KING	NEW YORK
KING	DALLAS
KING	CHICAGO
KING	BOSTON
MILLER	NEW YORK
MILLER	DALLAS

```
MILLER      CHICAGO
MILLER      BOSTON
```

Правильное результирующее множество должно выглядеть следующим образом:

```
ENAME      LOC
-----
CLARK      NEW YORK
KING       NEW YORK
MILLER     NEW YORK
```

РЕШЕНИЕ

Чтобы вернуть правильное результирующее множество, необходимо выполнить объединение таблиц в конструкции `FROM`:

```
1 select e.ename, d.loc
2    from emp e, dept d
3   where e.deptno = 10
4         and d.deptno = e.deptno
```

Обсуждение

Посмотрим на данные таблицы `DEPT`:

```
select * from dept
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Как можно видеть, отдел 10 находится в Нью-Йорке, и понятно, что результирующее множество, содержащее любой другой город, кроме Нью-Йорка, будет неправильным. Количество строк, возвращаемых неправильным запросом, представляет собой произведение кардинальностей (мощностей) двух таблиц в конструкции `FROM`. В этом запросе применяется фильтр для выбора из таблицы `EMP` только сотрудников отдела 10, в результате чего возвращаются три строки. Но поскольку для таблицы `DEPT` не применяется никакого фильтра, то из нее возвращаются все четыре строки. Умножив три на четыре, мы получаем двенадцать, поэтому неправильный запрос и возвращает двенадцать строк. Как правило, избежать декартова произведения можно, применяя правило $n-1$, где n — количество таблиц в конструкции `FROM`, а $n-1$ — минимальное количество объединений, необходимых для исключения декартова произведения. В зависимости от того, какие столбцы таблиц являются ключевыми и по каким столбцам выполняется объединение, вполне может потребоваться более чем $n-1$ объединений, но при создании запросов начать лучше всего с этого их количества.



При правильном использовании декартовы произведения могут быть полезными. Они часто задействуются, в том числе для транспонирования или сведения (или отмены сведения) результирующего множества, генерации последовательности значений и эмулирования цикла (хотя последние две задачи можно решить посредством рекурсивных обобщенных табличных выражений).

3.9. Выполнение объединений при использовании агрегатных функций

ЗАДАЧА

Требуется выполнить агрегирование, но ваш запрос охватывает несколько таблиц, поэтому необходимо убедиться, что объединения не нарушат агрегацию. Например, нужно вычислить суммы зарплат и премий всех служащих отдела 10. Но некоторые служащие получили несколько премий, и объединение таблицы EMP с таблицей EMP_BONUS, содержащей данные о премиях сотрудников для этой задачи, вызывает возвращение неправильных значений агрегатной функцией SUM. Пусть таблица EMP_BONUS имеет следующий вид:

```
select * from emp_bonus
```

EMPNO	RECEIVED	TYPE
7934	17-MAR-2005	1
7934	15-FEB-2005	2
7839	15-FEB-2005	3
7782	15-FEB-2005	1

Рассмотрим теперь следующий запрос, который возвращает данные о зарплате и премиях всех служащих отдела 10. Размер премии определяется по таблице BONUS.TYPE. Премия типа 1 составляет 10% зарплаты служащего, типа 2 — 20% и типа 3 — 30%.

```
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3
       end as bonus
from emp e, emp_bonus eb
where e.empno = eb.empno
and e.deptno = 10
```

EMPNO	ENAME	SAL	DEPTNO	BONUS
7934	MILLER	1300	10	130
7934	MILLER	1300	10	260
7839	KING	5000	10	1500
7782	CLARK	2450	10	245

Пока что все хорошо. Но при попытке присоединить таблицу EMP_BONUS, чтобы вычислить сумму премий, возникают проблемы:

```
select deptno,
       sum(sal) as total_sal,
       sum(bonus) as total_bonus
  from (
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3
          end as bonus
  from emp e, emp_bonus eb
 where e.empno = eb.empno
       and e.deptno = 10
       ) x
 group by deptno
```

DEPTNO	TOTAL_SAL	TOTAL_BONUS
10	10050	2135

Тогда как сумма премий TOTAL_BONUS вычисляется правильно, сумма зарплат TOTAL_SAL — нет. Правильная сумма зарплат для отдела 10 должна быть 8750, как показывает следующий запрос:

```
select sum(sal) from emp where deptno=10
```

SUM(SAL)
8750

Но почему сумма TOTAL_SAL вычисляется неправильно? Потому что объединение создает дубликаты строк в столбце SAL. Рассмотрим следующий запрос, который объединяет таблицы EMP и EMP_BONUS:

```
select e.ename,
       e.sal
  from emp e, emp_bonus eb
 where e.empno = eb.empno
       and e.deptno = 10
```

ENAME	SAL
CLARK	2450
KING	5000
MILLER	1300
MILLER	1300

Теперь можно легко видеть причину неправильного вычисления суммы TOTAL_SAL — зарплата служащего MILLER учитывается дважды. Правильное конечное результирующее множество должно выглядеть следующим образом:

DEPTNO	TOTAL_SAL	TOTAL_BONUS
10	8750	2135

РЕШЕНИЕ

Вычисление общих сумм в объединенных таблицах требует осторожного подхода. Избежать неправильных результатов вычислений, вызываемых дублированием строк при объединении таблиц с помощью агрегатных функций, можно двумя способами. Первый — просто используя в вызове агрегатной функции ключевое слово `DISTINCT`, обеспечивающее обработку только однозначных экземпляров каждого значения. Второй — выполняя агрегирование во вложенном запросе, прежде чем выполнять объединение. Это позволяет избежать неправильных вычислений, поскольку агрегация будет вычислена еще до объединения, что полностью устраняет проблему. Предлагаемые в этом разделе решения основаны на использовании ключевого слова `DISTINCT`. Метод с вычислением агрегации во вложенном запросе, выполняемом до объединения, рассматривается в разд. «Обсуждение».

MySQL и PostgreSQL

Суммируем только однозначные (`DISTINCT`) значения зарплаты:

```

1 select deptno,
2       sum(distinct sal) as total_sal,
3       sum(bonus) as total_bonus
4   from (
5 select e.empno,
6       e.ename,
7       e.sal,
8       e.deptno,
9       e.sal*case when eb.type = 1 then .1
10              when eb.type = 2 then .2
11              else .3
12              end as bonus
13   from emp e, emp_bonus eb
14   where e.empno = eb.empno
15         and e.deptno = 10
16         ) x
17  group by deptno

```

DB2, Oracle и SQL Server

Кроме предыдущего решения, эти платформы поддерживают еще и альтернативное решение с использованием оконной функции `SUM OVER`:

```

1 select distinct deptno,total_sal,total_bonus
2   from (
3 select e.empno,
4        e.ename,
5        sum(distinct e.sal) over
6          (partition by e.deptno) as total_sal,
7        e.deptno,
8        sum(e.sal*case when eb.type = 1 then .1
9                  when eb.type = 2 then .2
10                 else .3 end) over
11          (partition by deptno) as total_bonus
12   from emp e, emp_bonus eb
13  where e.empno = eb.empno
14        and e.deptno = 10
15        ) x

```

Обсуждение

MySQL and PostgreSQL

Второй запрос в *разд. «Задача»* этого рецепта выполняет объединение таблиц EMP и EMP_BONUS, возвращая две строки для служащего MILLER. Это обстоятельство и является причиной неправильного вычисления общей суммы зарплаты EMP.SAL (поскольку зарплата этого служащего учитывается дважды). Эта проблема решается простым суммированием только однозначных значений EMP.SAL, возвращенных запросом. Возможно и следующее альтернативное решение: сначала вычисляется общая сумма всех зарплат отдела 10, после чего эта строка объединяется с таблицей EMP, которая затем объединяется с таблицей EMP_BONUS.

Следующий запрос применим для всех рассматриваемых здесь СУБД:

```

select d.deptno,
       d.total_sal,
       sum(e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3 end) as total_bonus
  from emp e,
       emp_bonus eb,
  (
select deptno, sum(sal) as total_sal
  from emp
 where deptno = 10
 group by deptno
  ) d
 where e.deptno = d.deptno
       and e.empno = eb.empno
 group by d.deptno,d.total_sal

```

DEPTNO	TOTAL_SAL	TOTAL_BONUS
10	8750	2135

DB2, Oracle и SQL Server

Это альтернативное решение использует оконную функцию `SUM OVER`. Соответствующий запрос состоит из строк 3–14 рассмотренного ранее решения для этих СУБД и возвращает следующее результирующее множество:

```
select e.empno,
       e.ename,
       sum(distinct e.sal) over
         (partition by e.deptno) as total_sal,
       e.deptno,
       sum(e.sal*case when eb.type = 1 then .1
              when eb.type = 2 then .2
              else .3 end) over
         (partition by deptno) as total_bonus
from emp e, emp_bonus eb
where e.empno = eb.empno
      and e.deptno = 10
```

EMPNO	ENAME	TOTAL_SAL	DEPTNO	TOTAL_BONUS
7934	MILLER	8750	10	2135
7934	MILLER	8750	10	2135
7782	CLARK	8750	10	2135
7839	KING	8750	10	2135

Здесь оконная функция `SUM OVER` вызывается два раза. В первом вызове вычисляется сумма однозначных значений зарплат для заданного сегмента или группы. В нашем случае таким сегментом являются все строки, для которых значение столбца `DEPTNO` равно 10, а сумма однозначных значений зарплат для этого сегмента составляет 8750. Следующий вызов `SUM OVER` вычисляет сумму премий для этого же сегмента. Конечное результирующее множество создается выбором из этого результата однозначных значений `DEPTNO`, `TOTAL_SAL` и `TOTAL_BONUS`.

3.10. Выполнение внешних объединений при использовании агрегатных функций

ЗАДАЧА

Постановка задачи та же, что и для *рецепта 3.9*, с тем отличием, что не все служащие отдела 10 получили премии, что отражено должным образом в таблице `EMP_BONUS`:


```
select * from emp_bonus
```

EMPNO	RECEIVED	TYPE
7934	17-MAR-2005	1
7934	15-FEB-2005	2

По идее запрос для вычисления сумм всех зарплат и всех бонусов всех служащих отдела 10 должен выглядеть следующим образом:

```
select deptno,
       sum(sal) as total_sal,
       sum(bonus) as total_bonus
  from (
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3 end as bonus
  from emp e, emp_bonus eb
 where e.empno = eb.empno
       and e.deptno = 10
       )
 group by deptno
```

DEPTNO	TOTAL_SAL	TOTAL_BONUS
10	2600	390

Результат для суммы всех премий TOTAL_BONUS правильный, но для суммы всех зарплат TOTAL_SAL — нет. Неправильность результата TOTAL_SAL демонстрируется следующим запросом:

```
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3 end as bonus
  from emp e, emp_bonus eb
 where e.empno = eb.empno
       and e.deptno = 10
```

EMPNO	ENAME	SAL	DEPTNO	BONUS
7934	MILLER	1300	10	130
7934	MILLER	1300	10	260

Как можно видеть, вместо суммы всех зарплат служащих отдела 10 исходный запрос суммирует только значения зарплаты служащего MILLER, складывая его дважды. Правильное же результирующее множество должно быть следующим:

```
DEPTNO TOTAL_SAL TOTAL_BONUS
-----
10      8750      390
```

РЕШЕНИЕ

Решение аналогично решению задачи *рецепта 3.9*, но здесь выполняется внешнее объединение с таблицей EMP_BONUS, чтобы обеспечить включение всех служащих отдела 10.

DB2, MySQL, PostgreSQL и SQL Server

Выполняем внешнее объединение с таблицей EMP_BONUS, а затем суммируем только однозначные значения зарплат для отдела 10:

```
1 select deptno,
2     sum(distinct sal) as total_sal,
3     sum(bonus) as total_bonus
4   from (
5 select e.empno,
6     e.ename,
7     e.sal,
8     e.deptno,
9     e.sal*case when eb.type is null then 0
10            when eb.type = 1 then .1
11            when eb.type = 2 then .2
12            else .3 end as bonus
13   from emp e left outer join emp_bonus eb
14     on (e.empno = eb.empno)
15   where e.deptno = 10
16   )
17 group by deptno
```

Можно также использовать оконную функцию SUM OVER:

```
1 select distinct deptno,total_sal,total_bonus
2   from (
3 select e.empno,
4     e.ename,
5     sum(distinct e.sal) over
6     (partition by e.deptno) as total_sal,
7     e.deptno,
8     sum(e.sal*case when eb.type is null then 0
9            when eb.type = 1 then .1
10           when eb.type = 2 then .2
11           else .3
12           end) over
```

```

13         (partition by deptno) as total_bonus
14     from emp e left outer join emp_bonus eb
15         on (e.empno = eb.empno)
16     where e.deptno = 10
17         ) x

```

Обсуждение

Второй запрос в *разд. «Задача»* этого рецепта выполняет объединение таблиц EMP и EMP_BONUS, возвращая только строки для служащего MILLER. Это обстоятельство и является причиной неправильного вычисления общей суммы зарплаты EMP.SAL (поскольку для других служащих отдела 10 нет значений премий, и их зарплаты не включаются в сумму зарплат). Решение состоит в выполнении внешнего объединения таблицы EMP с таблицей EMP_BONUS, чтобы включить в результирующее множество даже служащих без премий. Для служащих без премии значение EMP_BONUS.TYPE будет NULL. Важно иметь в виду это обстоятельство, поскольку здесь выражение CASE слегка отличается от версии этого выражения в *рецепте 3.9*. Если значение EMP_BONUS.TYPE равно NULL, выражение CASE возвращает значение 0, которое не влияет на общую сумму.

Есть и альтернативное решение: сначала вычисляется общая сумма всех зарплат отдела 10, которая затем добавляется в таблицу EMP, после чего эта таблица, в свою очередь, объединяется с таблицей EMP_BONUS. Таким образом устраняется необходимость в выполнении внешнего объединения. Следующий запрос подерживается всеми СУБД, рассматриваемыми в этой книге:

```

select d.deptno,
       d.total_sal,
       sum(e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3 end) as total_bonus
  from emp e,
       emp_bonus eb,
  (
select deptno, sum(sal) as total_sal
  from emp
 where deptno = 10
 group by deptno
 ) d
 where e.deptno = d.deptno
       and e.empno = eb.empno
 group by d.deptno,d.total_sal

```

DEPTNO	TOTAL_SAL	TOTAL_BONUS
10	8750	390

3.11. Возвращение отсутствующих данных из нескольких таблиц

ЗАДАЧА

При работе одновременно с несколькими таблицами требуется вернуть данные, отсутствующие в какой-либо из них. Чтобы вернуть строки из таблицы DEPT, для которых нет соответствующих строк в таблице EMP (т. е. любой отдел, в котором нет служащих), нужно выполнить внешнее объединение. Попытаемся решить эту задачу посредством следующего запроса, который возвращает из таблицы DEPT все значения DEPTNO и DNAME вместе с именами всех служащих в каждом отделе (если отдел имеет служащих):

```
select d.deptno,d.dname,e.ename
       from dept d left outer join emp e
         on (d.deptno=e.deptno)
```

DEPTNO	DNAME	ENAME
20	RESEARCH	SMITH
30	SALES	ALLEN
30	SALES	WARD
20	RESEARCH	JONES
30	SALES	MARTIN
30	SALES	BLAKE
10	ACCOUNTING	CLARK
20	RESEARCH	SCOTT
10	ACCOUNTING	KING
30	SALES	TURNER
20	RESEARCH	ADAMS
30	SALES	JAMES
20	RESEARCH	FORD
10	ACCOUNTING	MILLER
40	OPERATIONS	

Возвращение последней строки отдела 40 (OPERATIONS), несмотря на отсутствие в нем служащих, обусловлено выполнением внешнего объединения таблицы EMP с таблицей DEPT. Теперь предположим, что таблица EMP содержит строку служащего, для которого нет значения в столбце DEPTNO. Каким образом можно вернуть предыдущее результирующее множество с дополнительной строкой для служащего без значения отдела? Иными словами, одним запросом нужно выполнить внешнее объединение как таблицы EMP с таблицей DEPT, так и таблицы DEPT с таблицей EMP.

Служащий без номера отдела добавляется в таблицу EMP следующим запросом:

```
insert into emp (empno,ename,job,mgr,hiredate,sal,comm,deptno)
select 1111,'YODA','JEDI',null,hiredate,sal,comm,null
       from emp
       where ename = 'KING'
```

А первая попытка решить задачу может быть в виде такого запроса:

```
select d.deptno,d.dname,e.ename
       from dept d right outer join emp e
       on (d.deptno=e.deptno)
```

DEPTNO	DNAME	ENAME
10	ACCOUNTING	MILLER
10	ACCOUNTING	KING
10	ACCOUNTING	CLARK
20	RESEARCH	FORD
20	RESEARCH	ADAMS
20	RESEARCH	SCOTT
20	RESEARCH	JONES
20	RESEARCH	SMITH
30	SALES	JAMES
30	SALES	TURNER
30	SALES	BLAKE
30	SALES	MARTIN
30	SALES	WARD
30	SALES	ALLEN
		YODA

Это внешнее объединение возвращает нового служащего, но при этом теряет отдел OPERATIONS из исходного результирующего множества. Правильное результирующее множество должно содержать как строку со служащим YODA, так и строку с отделением OPERATIONS:

DEPTNO	DNAME	ENAME
10	ACCOUNTING	CLARK
10	ACCOUNTING	KING
10	ACCOUNTING	MILLER
20	RESEARCH	ADAMS
20	RESEARCH	FORD
20	RESEARCH	JONES
20	RESEARCH	SCOTT
20	RESEARCH	SMITH
30	SALES	ALLEN
30	SALES	BLAKE
30	SALES	JAMES
30	SALES	MARTIN
30	SALES	TURNER
30	SALES	WARD
40	OPERATIONS	
		YODA

РЕШЕНИЕ

Возвратить данные, отсутствующие в какой-либо из обрабатываемых таблиц, по определенному общему значению можно с помощью полного внешнего объединения.

DB2, MySQL, PostgreSQL и SQL Server

Запрос с использованием команды явного полного внешнего объединения `FULL OUTER JOIN` возвращает строки, отсутствующие в какой-либо из обрабатываемых таблиц, а также все совпадающие строки:

```
1 select d.deptno,d.dname,e.ename
2     from dept d full outer join emp e
3     on (d.deptno=e.deptno)
```

Поскольку MySQL еще не поддерживает `FULL OUTER JOIN`, для этой СУБД выполняем по отдельности правое и левое внешние объединения, соединяя их результаты оператором `UNION`:

```
1 select d.deptno,d.dname,e.ename
2     from dept d right outer join emp e
3     on (d.deptno=e.deptno)
4 union
5 select d.deptno,d.dname,e.ename
6     from dept d left outer join emp e
7     on (d.deptno=e.deptno)
```

Oracle

Для этой СУБД можно использовать любое из предыдущих решений или же собственный синтаксис Oracle для внешнего объединения:

```
1 select d.deptno,d.dname,e.ename
2     from dept d, emp e
3     where d.deptno = e.deptno(+)
4 union
5 select d.deptno,d.dname,e.ename
6     from dept d, emp e
7     where d.deptno(+) = e.deptno
```

Обсуждение

Полное внешнее объединение — это просто комбинация двух типов внешних объединений (левого и правого). Чтобы увидеть «закулисные» подробности работы полного внешнего объединения, просто выполним каждый тип внешнего объединения по отдельности, а затем объединим их результаты с помощью оператора `UNION`. Следующий запрос выполняет левое внешнее объединение и возвращает строки из таблицы `DEPT` и все совпадающие строки из таблицы `EMP` (при наличии таковых):

```
select d.deptno,d.dname,e.ename
       from dept d left outer join emp e
       on (d.deptno = e.deptno)
```

DEPTNO	DNAME	ENAME
20	RESEARCH	SMITH
30	SALES	ALLEN
30	SALES	WARD
20	RESEARCH	JONES
30	SALES	MARTIN
30	SALES	BLAKE
10	ACCOUNTING	CLARK
20	RESEARCH	SCOTT
10	ACCOUNTING	KING
30	SALES	TURNER
20	RESEARCH	ADAMS
30	SALES	JAMES
20	RESEARCH	FORD
10	ACCOUNTING	MILLER
40	OPERATIONS	

А этот запрос выполняет правое внешнее объединение и возвращает строки из таблицы EMP и все совпадающие строки из таблицы DEPT (при наличии таковых):

```
select d.deptno,d.dname,e.ename
       from dept d right outer join emp e
       on (d.deptno = e.deptno)
```

DEPTNO	DNAME	ENAME
10	ACCOUNTING	MILLER
10	ACCOUNTING	KING
10	ACCOUNTING	CLARK
20	RESEARCH	FORD
20	RESEARCH	ADAMS
20	RESEARCH	SCOTT
20	RESEARCH	JONES
20	RESEARCH	SMITH
30	SALES	JAMES
30	SALES	TURNER
30	SALES	BLAKE
30	SALES	MARTIN
30	SALES	WARD
30	SALES	ALLEN
	YODA	

Результаты этих двух подзапросов объединяются оператором UNION, возвращая конечное результирующее множество.

3.12. Значения *NULL* в вычислениях и сравнениях

ЗАДАЧА

Значение *NULL* никогда не может быть равным или не равным любому значению, даже другому значению *NULL*. Но нам нужно выполнять операции со значениями столбца, который может содержать значения *NULL*, так же как и операции с действительными значениями. Например, нам нужно найти в таблице *EMP* всех служащих, для которых значение премии (*COMM*) меньше, чем размер премии служащего *WARD*. Результирующее множество также должно содержать служащих, для которых значение премии равно *NULL*.

РЕШЕНИЕ

С помощью функции *COALESCE* преобразовываем значение *NULL* в реальное значение, которое затем используем в обычных операциях сравнения:

```
1 select ename,comm
2    from emp
3  where coalesce(comm,0) < ( select comm
4                             from emp
5                             where ename = 'WARD' )
```

Обсуждение

Функция *COALESCE* возвращает первое значение не-*NULL* в списке переданных ей значений. При обнаружении значения *NULL* оно заменяется значением 0, которое затем сравнивается со значением премии служащего *WARD*. Это можно увидеть, поместив функцию *COALESCE* в список оператора *SELECT*:

```
select ename,comm,coalesce(comm,0)
    from emp
  where coalesce(comm,0) < ( select comm
                             from emp
                             where ename = 'WARD' )
```

ENAME	COMM	COALESCE (COMM, 0)
SMITH		0
ALLEN	300	300
JONES		0
BLAKE		0
CLARK		0
SCOTT		0
KING		0
TURNER	0	0
ADAMS		0

JAMES	0
FORD	0
MILLER	0

3.13. Подведем итоги

Операции объединения играют важную роль при обращении к базам данных с запросами — чтобы найти требуемые данные, вполне нормально объединить две или более таблиц. Овладев разными типами и комбинациями объединений, рассмотренными в этой главе, вы сможете успешно извлекать требуемые данные.

Вставка, обновление и удаление записей

В предыдущих главах мы рассмотрели основные методы работы с запросами, которые фокусировались на задаче извлечения данных из базы данных. В этой главе мы сменим направление и сосредоточимся на следующих трех предметных областях:

- ◆ добавление в базу данных новых записей (строк);
- ◆ обновление существующих записей;
- ◆ удаление ненужных записей.

Соответственно этому списку, сначала приводятся рецепты по вставке записей, затем по их обновлению и, наконец, по удалению.

Вставка записей обычно представляет собой простую задачу, состоящую в добавлении одной строки. Но для многих случаев более эффективным подходом является вставка набора записей. Исходя из этого, мы также рассмотрим методы для одновременной вставки сразу нескольких строк.

Аналогично, процедуры обновления и удаления записей мы начнем рассматривать с простых примеров удаления и обновления одной строки. Но обновлять можно целые множества записей одновременно, к тому же очень эффективными методами. Для удаления записей также существует много удобных способов. Например, строки из одной таблицы можно удалить в зависимости от наличия соответствующих строк в другой таблице.

А сравнительно новая возможность языка SQL даже позволяет вставлять, обновлять и удалять записи одновременно. Сейчас эта возможность может показаться вам не очень полезной, однако оператор MERGE представляет собой мощный способ синхронизации таблицы базы данных с внешним источником данных (например, с потоком неструктурированных файлов от удаленной системы). Этот оператор подробно рассматривается в *рецепте 4.11*.

4.1. Вставка новой записи

ЗАДАЧА

Требуется вставить новую запись (строку) в таблицу — например, добавить новый отдел в таблицу DEPT. Значение поля DEPTNO должно быть 50, поля DNAME — PROGRAMMING, а поля LOC — BALTIMORE.

РЕШЕНИЕ

Для вставки в таблицу одной строки используется оператор `INSERT`:

```
insert into dept (deptno,dname,loc)
values (50,'PROGRAMMING','BALTIMORE')
```

СУБД DB2, SQL Server, PostgreSQL и MySQL поддерживают вставку нескольких строк одновременно — при указании соответствующего значения их количества в списке `VALUES`:

```
/* вставка нескольких строк */
insert into dept (deptno,dname,loc)
values (1,'A','B'),
       (2,'B','C')
```

Обсуждение

Оператор `INSERT` создает в таблицах баз данных новые строки. Синтаксис оператора `INSERT` для вставки одной строки одинаков для всех рассматриваемых здесь баз данных.

Выражение можно упростить, опустив в нем список столбцов:

```
insert into dept
values (50,'PROGRAMMING','BALTIMORE')
```

Но при этом в списке `VALUES` обязательно нужно указать значения для всех столбцов таблицы в таком же порядке, в котором их значения предоставляются в ответе на запрос `SELECT *`. Обратите внимание — для *всех* столбцов, ибо в противном случае некоторым столбцам новой строки будут присвоены значения `NULL`. И если в такие столбцы вставка значений `NULL` не допускается, запрос на вставку завершится ошибкой.

4.2. Вставка значений по умолчанию

ЗАДАЧА

При создании таблицы она может быть определена так, чтобы принимать значения по умолчанию для конкретных ее столбцов. И вы хотите вставить строку значений по умолчанию, не указывая явно значения в списке `VALUES`. Рассмотрим следующую таблицу:

```
create table D (id integer default 0)
```

В эту таблицу вы хотите вставить ноль (0) без явного указания нуля в списке значений `VALUES` оператора `INSERT`. Попросту говоря, вы собираетесь явно вставить в таблицу значение по умолчанию, каким бы оно ни было.

РЕШЕНИЕ

Все рассматриваемые в этой книге СУБД поддерживают использование ключевого слова `DEFAULT` для задания значения по умолчанию столбцам вставляемой строки. Некоторые другие СУБД для решения этой задачи предоставляют и иные способы.

Далее приводится пример использования ключевого слова `DEFAULT` с оператором `INSERT`:

```
insert into D values (default)
```

Вы также можете явно указать имя столбца, и это нужно делать всегда, когда вы вставляете данные не во все столбцы таблицы.

```
insert into D (id) values (default)
```

СУБД Oracle8i и предыдущие ее версии не поддерживают ключевое слово `DEFAULT`. Возможность явного указания значения столбца по умолчанию появилась только в версии Oracle9i.

В MySQL, если для всех столбцов таблицы было определено значение по умолчанию, допускается задание пустого списка значений столбцов:

```
insert into D values ()
```

При этом всем столбцам присваивается соответствующее значение по умолчанию.

В СУБД PostgreSQL и SQL Server поддерживается оператор `DEFAULT VALUES`:

```
insert into D default values
```

Оператор `DEFAULT VALUES` присваивает всем столбцам соответствующие значения по умолчанию.

Обсуждение

Ключевое слово `DEFAULT` в списке `VALUES` обеспечивает вставку в столбец значения по умолчанию, заданного для этого столбца при создании таблицы. Это ключевое слово поддерживается всеми рассматриваемыми здесь СУБД.

СУБД MySQL, PostgreSQL и SQL Server предоставляют дополнительную опцию, если для всех столбцов таблицы определено значение по умолчанию (как для таблицы D в нашем случае). В частности, новую строку со значениями по умолчанию для всех столбцов можно создать, используя пустой список `VALUES` (MySQL) или оператор `DEFAULT VALUES` (PostgreSQL и SQL Server). В противном случае ключевое слово `DEFAULT` нужно указывать для каждого столбца таблицы.

Для таблиц, в которых значения по умолчанию определены не для всех столбцов, значения по умолчанию задаются, просто не указывая соответствующий столбец в списке `INSERT`. Использовать ключевое слово `DEFAULT` при этом не требуется. Например, предположим, что мы создали таблицу D с дополнительным столбцом, для которого не задали значение по умолчанию:

```
create table D (id integer default 0, foo varchar(10))
```

Вставить в эту таблицу строку со значением по умолчанию для столбца `ID` можно, указав в списке `INSERT` название только столбца `FOO`:

```
insert into D (foo) values ('Bar')
```

В результате значение столбца `ID` вставленной строки будет `0`, а столбца `FOO` — `Bar`. Столбцу `ID` присваивается значение по умолчанию, поскольку не было указано никакого другого значения.

4.3. Переопределение значения по умолчанию значением `NULL`

ЗАДАЧА

При вставке строки со столбцом, имеющим значение по умолчанию, требуется переопределить это значение значением `NULL`. Возьмем, например, следующую таблицу:

```
create table D (id integer default 0, foo VARCHAR(10))
```

В эту таблицу нужно вставить строку, значение столбца `ID` которой будет `NULL`.

РЕШЕНИЕ

Значение `NULL` для столбца `ID` можно явно задать в списке значений:

```
insert into d (id, foo) values (null, 'Brighten')
```

Обсуждение

Многие пользователи не знают, что в списке `VALUES` оператора `INSERT` для столбцов можно явно задавать значение `NULL`. Обычно, если столбцу по какой-либо причине не задается значение, его просто не указывают в списке столбцов и значений:

```
insert into d (foo) values ('Brighten')
```

В этом примере для столбца `ID` не указано никакого значения. Можно было бы ожидать, что столбцу будет присвоено значение `NULL`, но, увы, поскольку при создании таблицы для столбца `ID` было задано значение по умолчанию, в результате исполнения этого запроса ему присваивается значение `0` (т. е. значение по умолчанию). Столбцу можно присвоить значение `NULL`, невзирая на его значение по умолчанию, указав для него в запросе `INSERT` значение `NULL` (за исключением случая, когда для столбца определено ограничение, запрещающее для него значение `NULL`).

4.4. Копирование строк одной таблицы в другую

ЗАДАЧА

Требуется с помощью запроса скопировать строки из одной таблицы в другую. Уровень сложности запроса не имеет значения — только его способность спра-

виться с поставленной задачей. Например, нужно скопировать строки из таблицы DEPT в таблицу DEPT_EAST, которая имеет такую же структуру, что и таблица DEPT (такие же столбцы таких же типов), и не содержит никаких строк.

РЕШЕНИЕ

Задача решается с помощью оператора INSERT, требуемые строки для которого выбираются посредством запроса SELECT:

```
1 insert into dept_east (deptno,dname,loc)
2 select deptno,dname,loc
3   from dept
4  where loc in ( 'NEW YORK','BOSTON' )
```

Обсуждение

Требуемые строки из исходной таблицы выбираются с помощью запроса SELECT, следующего за оператором INSERT. Чтобы выбрать все строки, просто опускаем из запроса блок WHERE. Аналогично обычной вставке строк, явно указывать столбцы, в которые вставляются значения, необязательно. Но в таком случае данные необходимо вставить во все столбцы таблицы, указывая значения в списке SELECT в соответствующем порядке, как показано в *рецепте 4.1*.

4.5. Копирование определения таблицы

ЗАДАЧА

Требуется создать новую таблицу с таким же набором столбцов, как и в существующей таблице. Например, создать копию таблицы DEPT, присвоив ей название DEPT_2. Копировать строки исходной таблицы в новую не нужно — только структуру ее столбцов.

РЕШЕНИЕ

DB2

Используем оператор LIKE с командой CREATE TABLE:

```
create table dept_2 like dept
```

Oracle, MySQL и PostgreSQL

Используем команду CREATE TABLE с подзапросом, который не возвращает никаких строк:

```
1 create table dept_2
2 as
3 select *
```

```
4 from dept
5 where 1 = 0
```

SQL Server

Используем оператор `INTO` в запросе, который не возвращает никаких строк:

```
1 select *
2 into dept_2
3 from dept
4 where 1 = 0
```

Обсуждение

DB2

Для этой СУБД создать таблицу по шаблону другой таблицы можно посредством команды `CREATE TABLE...LIKE`, просто указав название таблицы-шаблона после ключевого слова `LIKE`.

Oracle, MySQL и PostgreSQL

Запрос `Create Table As Select (CTAS)` создает новую таблицу, при этом заполняя ее строками из вашего запроса, — если вы не укажете ложное (`false`) условие в выражении `WHERE`. В таком случае строки не переносятся и создается пустая таблица. В приведенном для этой группы СУБД решении в выражении `WHERE` условие `1 = 0` является ложным. Соответственно, в результате запрос `CTAS` создает пустую таблицу по шаблону столбцов, указанных в операторе `SELECT` запроса.

SQL Server

При использовании в запросе `SELECT` ключевого слова `INTO` создается новая таблица, которая заполняется возвращенными запросом строками. Но если условие в выражении `WHERE` ложное, то строки не возвращаются, и создается пустая таблица. В приведенном решении в выражении `WHERE` условие `1 = 0` является ложным. В результате запрос создает пустую таблицу по шаблону столбцов, указанных в операторе `SELECT` запроса.

4.6. Вставка строк одновременно в несколько таблиц

ЗАДАЧА

Возвращенные запросом строки требуется вставить одновременно в несколько таблиц. Например, строки из таблицы `DEPT` нужно вставить в таблицы `DEPT_EAST`, `DEPT_WEST` и `DEPT_MID`. Все таблицы имеют такую же структуру (такие же столбцы с такими же типами данных), что и таблица `DEPT`, и в настоящий момент не содержат никаких строк.

РЕШЕНИЕ

Решение заключается во вставке строк результирующего множества запроса в таблицы назначения. Эта задача отличается от задачи в *рецепте 4.4* тем, что здесь строки вставляются в несколько таблиц назначения.

Oracle

Используем выражение `INSERT ALL` или `INSERT FIRST`. Оба эти выражения имеют одинаковый синтаксис, за исключением варианта выбора ключевого слова `ALL` или `FIRST`. В следующем запросе используется выражение `INSERT ALL`, обеспечивающее вставку выбранных строк во все возможные таблицы назначения:

```

1 insert all
2   when loc in ('NEW YORK','BOSTON') then
3     into dept_east (deptno,dname,loc) values (deptno,dname,loc)
4   when loc = 'CHICAGO' then
5     into dept_mid (deptno,dname,loc) values (deptno,dname,loc)
6   else
7     into dept_west (deptno,dname,loc) values (deptno,dname,loc)
8   select deptno,dname,loc
9   from dept

```

DB2

Вставка осуществляется во вложенный запрос, который выполняет объединение `UNION ALL` всех таблиц назначения. При этом на таблицы необходимо наложить ограничения, обеспечивающие вставку строк в соответствующие таблицы:

```

create table dept_east
( deptno integer,
  dname varchar(10),
  loc varchar(10) check (loc in ('NEW YORK','BOSTON'))

```

```

create table dept_mid
( deptno integer,
  dname varchar(10),
  loc varchar(10) check (loc = 'CHICAGO'))

```

```

create table dept_west
( deptno integer,
  dname varchar(10),
  loc varchar(10) check (loc = 'DALLAS'))

```

```

1 insert into (
2   select * from dept_west union all
3   select * from dept_east union all
4   select * from dept_mid
5 ) select * from dept

```


MySQL, PostgreSQL и SQL Server

На момент подготовки этой книги указанные СУБД не поддерживают одновременную вставку в несколько таблиц.

Обсуждение

Oracle

В решении для этой СУБД используются выражения `WHEN...THEN...ELSE`, которые обрабатывают строки, возвращаемые вложенным запросом `SELECT`, обеспечивая их вставку в соответствующие таблицы. Хотя в приведенном примере оба выражения: и `INSERT ALL`, и `INSERT FIRST` — дали бы одинаковый результат, работают они по-разному. Выражение `INSERT FIRST` выходит из блока `WHEN...THEN...ELSE` после первого же выполненного условия, тогда как выражение `INSERT ALL` будет продолжать обработку всех условий даже после выполнения предыдущих условий. Таким образом, выражение `INSERT ALL` позволяет вставить одну и ту же строку в несколько таблиц.

DB2

Решение для DB2 несколько топорное. В нем на таблицы назначения требуется наложить ограничения, обеспечивающие вставку каждой обработанной в подзапросе строки в соответствующую таблицу. Подход состоит в осуществлении вставки строк в представление, определенное в качестве объединения всех таблиц (`UNION ALL`). Если на таблицы в операторе `INSERT` не наложены индивидуальные ограничения (т. е. к нескольким таблицам применено одно и то же ограничение), оператор не сможет определить, в какую таблицу вставлять строки, и его исполнение завершится ошибкой.

MySQL, PostgreSQL и SQL Server

На момент подготовки этой книги указанные СУБД не предоставляют механизмов для вставки возвращаемых запросом строк в одну или несколько таблиц одним выражением.

4.7. Блокировка вставки данных в определенные столбцы

ЗАДАЧА

Требуется предотвратить вставку (пользователями или программами) значений в определенные столбцы таблицы. Например, нужно разрешить вставку значений программой только в столбцы `EMPNO`, `ENAME` и `JOB` таблицы `EMP`.

РЕШЕНИЕ

Создаем представление таблицы, содержащее только столбцы, в которые разрешена вставка, и позволяем выполнять все вставки только через это представление.

Например, создадим представление, содержащее требуемые три столбца таблицы EMP:

```
create view new_emps as
select empno, ename, job
from emp
```

Пользователям и/или программам, которым разрешено заполнять эти три столбца, предоставляем доступ к созданному представлению, но не к самой таблице EMP. Тогда они смогут вставлять новые записи в таблицу EMP, создавая записи в представлении NEW_EMP, но при этом заполняя только те три столбца таблицы, которые указаны в определении представления.

Обсуждение

При вставке строк в простое представление, как в этом решении, сервер баз данных переносит их в базовую таблицу. Например, следующая операция вставки:

```
insert into new_emps
(empno ename, job)
values (1, 'Jonathan', 'Editor')
```

преобразовывается сервером в эту:

```
insert into emp
(empno ename, job)
values (1, 'Jonathan', 'Editor')
```

Также возможен альтернативный, но, вероятно, менее практичный подход со вставкой строк во вложенное представление:

```
insert into
(select empno, ename, job
from emp)
values (1, 'Jonathan', 'Editor')
```

Однако в настоящее время этот подход поддерживается только в Oracle.

Вставка строк посредством представления — тема весьма сложная. Правила ее выполнения с ростом сложности представлений усложняются очень быстро. Чтобы успешно применять возможность вставки посредством представлений, важно изучить документацию СУБД по этой теме и досконально в ней разобраться.

4.8. Изменение записей в таблице

ЗАДАЧА

Требуется изменить значения некоторых столбцов таблицы. Например, надо повысить зарплату служащих отдела 20 на 10%. В следующем результирующем множестве отображены данные столбцов DEPTNO, ENAME и SAL для всех служащих этого отдела:

```
select deptno,ename,sal
  from emp
 where deptno = 20
 order by 1,3
```

DEPTNO	ENAME	SAL
20	SMITH	800
20	ADAMS	1100
20	JONES	2975
20	SCOTT	3000
20	FORD	3000

Нам нужно повысить все значения SAL на 10%.

РЕШЕНИЕ

Для изменения значений существующих строк таблицы используется оператор UPDATE. Например:

```
1 update emp
2   set sal = sal*1.10
3   where deptno = 20
```

Обсуждение

Для изменения значений строк используется оператор UPDATE, а строки, подлежащие обновлению, указываются посредством предиката WHERE. При отсутствии этого предиката обновляются все строки таблицы. Выражение SAL*1.10 в приведенном решении повышает значение SAL на 10%.

В случае масштабного обновления весьма полезно предварительно просмотреть, какими будут его результаты. Это можно осуществить с помощью оператора SELECT, содержащего выражения, которые планируется использовать в выражениях SET. Например, следующий оператор SELECT отображает результаты повышения зарплаты на 10%:

```
select deptno,
       ename,
       sal as orig_sal,
       sal*1.10 as amt_to_add,
       sal*1.10 as new_sal
  from emp
 where deptno=20
 order by 1,5
```

DEPTNO	ENAME	ORIG_SAL	AMT_TO_ADD	NEW_SAL
20	SMITH	800	80	880
20	ADAMS	1100	110	1210

20 JONES	2975	298	3273
20 SCOTT	3000	300	3300
20 FORD	3000	300	3300

Повышение зарплаты разделено по двум столбцам: в одном отображается сумма повышения, а в другом — новая зарплата.

4.9. Обновление при условии наличия соответствующих строк

ЗАДАЧА

Требуется обновить строки в одной таблице при условии наличия соответствующих строк в другой таблице. Например, если таблица EMP_BONUS содержит запись о служащем, этому служащему нужно повысить зарплату на 20% в таблице EMP. Текущие данные таблицы EMP_BONUS отображены в следующем результирующем множестве:

```
select empno, ename
from emp_bonus
```

```
EMPNO ENAME
-----
7369 SMITH
7900 JAMES
7934 MILLER
```

РЕШЕНИЕ

В предикате WHERE оператора UPDATE размещаем подзапрос, чтобы найти в таблице EMP записи служащих, для которых также есть записи в таблице EMP_BONUS. В результате оператор UPDATE будет действовать только на возвращенные этим подзапросом строки, позволяя повысить на 20% зарплату соответствующим служащим:

```
1 update emp
2   set sal=sal*1.20
3  where empno in ( select empno from emp_bonus )
```

Обсуждение

Подзапрос возвращает строки, которые будут обновлены в таблице EMP. В предикате IN выполняется проверка на наличие значений столбца EMPNO таблицы EMP в списке значений EMPNO, возвращенного подзапросом. При обнаружении совпадающего значения обновляется соответствующее значение SAL.

Альтернативно, вместо IN можно использовать EXISTS:

```
update emp
   set sal= sal*1.20
```

```

where exists ( select null
               from emp_bonus
               where emp.empno=emp_bonus.empno )

```

Наличие значения NULL в списке SELECT подзапроса EXISTS может вызвать у вас удивление. Не озабочивайтесь, это значение не оказывает никакого отрицательного воздействия на обновление. Возможно, значение NULL делает код более понятным, подчеркивая тот факт, что, в отличие от решения с использованием подзапроса с оператором IN, в рассматриваемом случае выбор строк для обновления управляется предикатом WHERE подзапроса, а не значениями, возвращенными в результате обработки списка SELECT подзапроса.

4.10. Обновление значениями из другой таблицы

ЗАДАЧА

Требуется обновить строки одной таблицы значениями из другой. Например, имеется таблица NEW_SAL, в которой хранятся новые значения зарплат для ряда служащих. Содержимое этой таблицы следующее:

```

select *
  from new_sal

```

```

DEPTNO      SAL
-----
10          4000

```

Столбец DEPTNO является первичным ключом таблицы. Нам нужно обновить значения зарплат и премий служащих в таблице EMP, для которых значение EMP.DPTNO совпадает со значением NEW_SAL.DEPTNO. В частности, значение EMP.SAL нужно обновить значением NEW_SAL.SAL, а значение EMP.COMM обновить до 50% от значения NEW_SAL.SAL. Содержимое таблицы EMP следующее:

```

select deptno, ename, sal, comm
  from emp
 order by 1

```

```

DEPTNO ENAME      SAL      COMM
-----
10 CLARK      2450
10 KING      5000
10 MILLER    1300
20 SMITH     800
20 ADAMS     1100
20 FORD      3000
20 SCOTT     3000
20 JONES     2975
30 ALLEN     1600      300
30 BLAKE     2850
30 MARTIN    1250     1400

```

30 JAMES	950	
30 TURNER	1500	0
30 WARD	1250	500

РЕШЕНИЕ

Новые значения `COMM` находим и возвращаем в оператор `UPDATE` посредством объединения таблиц `NEW_SAL` и `EMP`. Обновления такого типа часто осуществляются с помощью связанного подзапроса или использованием обобщенного табличного выражения. Еще один подход состоит в создании представления (обычного или вложенного, в зависимости от того, какой тип поддерживается используемой базой данных) с последующим его обновлением.

DB2

Новые значения `SAL` и `COMM` в таблице задаем, используя связанный подзапрос. Определение строк таблицы `EMP`, подлежащих обновлению, также осуществляем с помощью связанного подзапроса.

```

1 update emp e set (e.sal,e.comm) = (select ns.sal, ns.sal/2
2                                 from new_sal ns
3                                 where ns.deptno=e.deptno)
4 where exists ( select *
5               from new_sal ns
6               where ns.deptno = e.deptno )

```

MySQL

Размещаем обе таблицы: `EMP` и `NEW_SAL`, в операторе `UPDATE` и объединяем в выражении `WHERE`:

```

1 update emp e, new_sal ns
2 set e.sal=ns.sal,
3 e.comm=ns.sal/2
4 where e.deptno=ns.deptno

```

Oracle

Для этой СУБД можно использовать или решение для `DB2`, или, альтернативно, выполнить обновление через вложенное представление:

```

1 update (
2   select e.sal as emp_sal, e.comm as emp_comm,
3          ns.sal as ns_sal, ns.sal/2 as ns_comm
4   from emp e, new_sal ns
5   where e.deptno = ns.deptno
6 ) set emp_sal = ns_sal, emp_comm = ns_comm

```

PostgreSQL

Решение, приведенное для DB2, также применимо и для PostgreSQL, но объединение можно также выполнить (что весьма удобно) непосредственно в операторе UPDATE:

```
1 update emp
2   set sal = ns.sal,
3     comm = ns.sal/2
4   from new_sal ns
5  where ns.deptno = emp.deptno
```

SQL Server

Решение, приведенное для DB2, также применимо и для SQL Server, но объединение можно также выполнить (как и в решении для PostgreSQL) непосредственно в операторе UPDATE:

```
1 update e
2   set e.sal = ns.sal,
3     e.comm = ns.sal/2
4   from emp e,
5     new_sal ns
6  where ns.deptno = e.deptno
```

Обсуждение

Прежде чем обсуждать приведенные решения, следует упомянуть один важный момент, касающийся обновлений, в которых новые значения предоставляются запросами. Блок WHERE в подзапросе связанного обновления не имеет ничего общего с блоком WHERE, применяемым к обновляемой таблице. Рассмотрим выражение UPDATE в разд. «Решение». Можно видеть, что объединение таблиц EMP и NEW_SAL выполняется по столбцу DEPTNO, результирующие строки которого возвращаются в выражение SET выражения UPDATE. Для служащих отдела 10 (DEPTNO = 10) возвращаются действительные значения, т. к. таблица NEW_SAL содержит соответствующие им значения DEPTNO. Но как быть со служащими других отделов? Поскольку таблица NEW_SAL не содержит значений DEPTNO для других отделов, то столбцам SAL и COMM для служащих отделов 20 и 30 (DEPTNO = 20 и 30) присваивается значение NULL. Если используемая СУБД не поддерживает какого-либо механизма для ограничения количества строк в результирующем множестве (например, функции LIMIT или TOP), сделать это в SQL можно только посредством использования блока WHERE. Чтобы обеспечить правильное выполнение рассматриваемого выражения UPDATE, нужно использовать блок WHERE для обновляемой таблицы совместно с блоком WHERE в связанном подзапросе.

DB2

Чтобы не допустить обновления всех строк в таблице EMP, нужно включить связанный подзапрос в блок WHERE выражения UPDATE. Одного выполнения объединения

(связанный подзапрос) в выражении SET будет недостаточно. Наличие блока WHERE в выражении UPDATE обеспечивает обновление только тех строк в таблице EMP, для значения DEPTNO которых есть совпадающее значение в таблице NEW_SAL. Это касается все рассматриваемых здесь СУБД.

Oracle

В решении Oracle с обновлением объединенного представления строки для обновления определяются посредством эквиобъединений. Проверить, какие строки будут обновлены, можно, выполнив запрос отдельно. Чтобы успешно выполнять обновления такого типа, необходимо понимать принцип сохранения ключей. Столбец DEPTNO таблицы NEW_SAL является ее первичным ключом. Следовательно, в пределах этой таблицы все значения столбца DEPTNO уникальны. Однако в результирующем множестве объединения таблиц EMP и NEW_SAL значения NEW_SAL.DEPTNO более не уникальны:

```
select e.empno, e.deptno e_dept, ns.sal, ns.deptno ns_deptno
       from emp e, new_sal ns
       where e.deptno = ns.deptno
```

EMPNO	E_DEPT	SAL	NS_DEPTNO
7782	10	4000	10
7839	10	4000	10
7934	10	4000	10

Чтобы Oracle могла выполнить это объединение, одна из таблиц должна быть *защищенной по ключу* (key-preserved), а это означает, что если ее значения не уникальны в результирующем множестве, они должны быть, по крайней мере, уникальны в исходной таблице. В нашем случае первичным ключом таблицы NEW_SAL является столбец DEPTNO, что, как упоминалось ранее, делает его значения уникальными в пределах этой таблицы. Поскольку значения столбца DEPTNO уникальны в исходной таблице, в результирующем множестве они могут повторяться, но при этом исходная таблица все равно остается защищенной по ключу, что позволяет успешно выполнять обновление.

PostgreSQL, SQL Server и MySQL

Синтаксис решений для этих СУБД несколько различается, но базовый принцип остается тем же. Возможность выполнять объединение непосредственно в выражении UPDATE чрезвычайно удобна. Явное указание (после ключевого слова UPDATE) таблицы, подлежащей обновлению, устраняет возможность путаницы в том, строки какой таблицы изменяются. Кроме этого, размещение объединений в блоке обновления (поскольку здесь присутствует явный блок WHERE) позволяет избежать некоторых ошибок при создании обновлений, использующих связанные подзапросы. В частности, если здесь пропустить объединение, то наличие проблемы станет очевидным.

4.11. Слияние записей

ЗАДАЧА

Требуется выполнить вставку, обновление или удаление записи в таблице в зависимости от наличия или состояния соответствующей записи. В частности, если запись существует, обновить ее, если нет — вставить, а если обновленная запись не отвечает определенному условию — удалить ее. Например, модифицировать таблицу EMP_COMMISSION следующим образом:

- ◆ если служащий из таблицы EMP_COMMISSION также присутствует в таблице EMP, обновляем его комиссионные COMM до значения 1000;
- ◆ удаляем из таблицы EMP_COMMISSION всех служащих, чьи комиссионные COMM могут быть обновлены до значения 1000, если их зарплата SAL меньше 2000;
- ◆ в противном случае вставляем значения EMPNO, ENAME и DEPTNO из таблицы EMP в таблицу EMP_COMMISSION.

По сути, в зависимости от наличия или отсутствия в таблице EMP_COMMISSION записи, совпадающей с текущей записью в таблице EMP, нам нужно или обновить (UPDATE), или вставить (INSERT) эту запись в таблицу EMP_COMMISSION. Но если значение комиссионных COMM обновленной записи превышает заданный предел, эту запись нужно удалить (DELETE).

Далее приводится текущее содержимое таблиц EMP и EMP_COMMISSION, соответственно:

```
select deptno, empno, ename, comm
  from emp
 order by 1
```

DEPTNO	EMPNO	ENAME	COMM
10	7782	CLARK	
10	7839	KING	
10	7934	MILLER	
20	7369	SMITH	
20	7876	ADAMS	
20	7902	FORD	
20	7788	SCOTT	
20	7566	JONES	
30	7499	ALLEN	300
30	7698	BLAKE	
30	7654	MARTIN	1400
30	7900	JAMES	
30	7844	TURNER	0
30	7521	WARD	500

```
select deptno, empno, ename, comm
  from emp_commission
 order by 1
```

DEPTNO	EMPNO	ENAME	COMM
10	7782	CLARK	
10	7839	KING	
10	7934	MILLER	

РЕШЕНИЕ

Для решения задач такого типа предназначено выражение `MERGE`, которое может по необходимости выполнять или обновление (`UPDATE`), или вставку (`INSERT`). Например:

```

1 merge into emp_commission ec
2 using (select * from emp) emp
3   on (ec.empno=emp.empno)
4   when matched then
5       update set ec.comm = 1000
6       delete where (sal < 2000)
7   when not matched then
8       insert (ec.empno,ec.ename,ec.deptno,ec.comm)
9       values (emp.empno,emp.ename,emp.deptno,emp.comm)

```

На момент подготовки этой книги только MySQL не поддерживает выражение `MERGE`. Все остальные рассматриваемые здесь СУБД, а также и многие другие должны работать с таким запросом.

Обсуждение

Операция объединения в строке 3 решения определяет, какие строки таблицы уже существуют и будут обновлены. Объединяются таблица `EMP_COMMISSION` (под псевдонимом `EC`) и результирующее множество подзапроса (под псевдонимом `EMP`). В случае успешного объединения две задействованные в нем строки считаются «совпавшими», и в блоке `WHEN MATCHED` выполняется обновление (`UPDATE`). В противном случае совпадения нет, и в блоке `WHEN NOT MATCHED` выполняется вставка (`INSERT`). Таким образом, строки из таблицы `EMP`, для которых в таблице `EMP_COMMISSION` нет строк с совпадающим значением столбца `EMPNO`, будут вставлены в эту таблицу. Из всех служащих в таблице `EMP` значения `COMM` в таблице `EMP_COMMISSION` должны обновиться только для служащих отдела 10 (`DEPTNO = 10`), а строки для остальных служащих должны вставиться. Кроме этого, т. к. служащий `MILLER` относится к отделу 10, значение `COMM` для него должно было бы обновиться, но, поскольку его значение `SAL` меньше 2000, запись о нем удаляется (`DELETE`) из таблицы `EMP_COMMISSION`.

4.12. Удаление всех записей таблицы

ЗАДАЧА

Требуется удалить из таблицы все записи.

РЕШЕНИЕ

Для удаления записей из таблиц служит команда `DELETE`. Например, следующая команда удаляет все записи из таблицы EMP:

```
delete from emp
```

Обсуждение

Команда `DELETE` без предиката `WHERE` удаляет все строки из указанной таблицы. Иногда для этого предпочтительней использовать команду `TRUNCATE`, которая не требует предиката `WHERE`, поскольку она работает быстрее. Но, по крайней мере, в Oracle отменить результаты исполнения команды `TRUNCATE` невозможно. Поэтому следует внимательно изучить документацию поставщика используемой СУБД на предмет различий в работе и отмене результатов команд `TRUNCATE` и `DELETE`.

4.13. Удаление определенных записей

ЗАДАЧА

Требуется удалить из таблицы записи, отвечающие определенному критерию.

РЕШЕНИЕ

Используем команду `DELETE`, указывая критерий для удаления в предикате `WHERE`. Например, следующая команда удаляет записи для всех служащих отдела 10:

```
delete from emp where deptno = 10
```

Обсуждение

Использование предиката `WHERE` с командой `DELETE` позволяет удалять не все, а только определенное подмножество строк таблицы. Поскольку даже в простых ситуациях можно удалить не те данные, что подразумевалось, то, прежде чем удалять строки, не забудьте проверить эффект условия удаления, задаваемого в предикате `WHERE`, указав его в выражении `SELECT`. Например, в приведенном примере в результате простой опечатки могли бы быть удалены сотрудники отдела 20 вместо 10.

4.14. Удаление одной записи

ЗАДАЧА

Требуется удалить из таблицы одну запись.

РЕШЕНИЕ

Это особый случай *рецепта 4.13*. Его особенность состоит в необходимости обеспечения достаточно узкого критерия выбора, чтобы указать для удаления только

одну запись. Часто в качестве такого критерия используют первичный ключ. Например, следующий запрос удаляет запись для служащего CLARK (EMPNO = 7782):

```
delete from emp where empno = 7782
```

Обсуждение

Ключевой момент удаления строк — определение строк для удаления, и воздействие команды DELETE всегда сводится к ее предикату WHERE. При отсутствии WHERE воздействие DELETE распространяется на все строки таблицы. Указывая условия удаления в предикате WHERE, можно сузить эту область к группе записей или даже к одной записи. Обычно если удаляется одна запись, она определяется по своему первичному ключу или по одному из ее уникальных ключей.



Критерий удаления на основе первичного или уникального ключа обеспечивает удаление только одной записи (это объясняется тем, что используемая СУБД не допустит наличия двух строк с одинаковым значением первичного или уникального ключа). В противном случае рекомендуется сначала проверить выбор правильных строк для удаления, чтобы убедиться в том, что случайно не будет удалено больше строк, чем подразумевалось.

4.15. Удаление строк, нарушающих ссылочную целостность

ЗАДАЧА

Из таблицы требуется удалить записи, ссылающиеся на несуществующие записи в какой-либо другой таблице. Например, для некоторых служащих может быть указан несуществующий номер отдела. Записи таких служащих нужно удалить.

РЕШЕНИЕ

Проверку действительности номеров отделов выполняем посредством предиката NOT EXISTS с подзапросом:

```
delete from emp
  where not exists (
    select * from dept
      where dept.deptno = emp.deptno
  )
```

Альтернативно, можно использовать запрос с предикатом NOT IN:

```
delete from emp
  where deptno not in (select deptno from dept)
```

Обсуждение

По сути, главное в удалении — это выбор, т. е. создание правильных условий предиката WHERE, определяющих записи для удаления.

В решении с предикатом `NOT EXISTS` проверка на наличие в таблице `DEPT` записи со значением `DEPTNO`, совпадающим с `DEPTNO` заданной записи в таблице `EMP`, осуществляется посредством связанного подзапроса. При положительном результате проверки запись в таблице `EMP` сохраняется. В противном случае запись удаляется. На это условие проверяются все записи в таблице `EMP`.

В решении с предикатом `IN` посредством подзапроса формируется список действительных номеров отделов, по которому затем проверяется значение `DEPTNO` каждой записи в таблице `EMP`. При обнаружении в таблице `EMP` записи со значением `DEPTNO`, отсутствующим в списке, она удаляется.

4.16. Удаление дубликатов записей

ЗАДАЧА

Требуется удалить из таблицы повторяющиеся записи. Возьмем, например, следующую таблицу:

```
create table dupes (id integer, name varchar(10))
```

```
insert into dupes values (1, 'NAPOLEON')
insert into dupes values (2, 'DYNAMITE')
insert into dupes values (3, 'DYNAMITE')
insert into dupes values (4, 'SHE SELLS')
insert into dupes values (5, 'SEA SHELLS')
insert into dupes values (6, 'SEA SHELLS')
insert into dupes values (7, 'SEA SHELLS')
```

```
select * from dupes order by 1
```

```

ID NAME
-----
 1 NAPOLEON
 2 DYNAMITE
 3 DYNAMITE
 4 SHE SELLS
 5 SEA SHELLS
 6 SEA SHELLS
 7 SEA SHELLS
```

Из каждой группы повторяющихся названий `NAME` — например, `SEA SHELLS`, нужно оставить одну запись с любым значением `ID` и удалить все остальные. То есть для той же группы записей `SEA SHELLS` нам безразлично, какие записи удалить: 5 и 6, 5 и 7 или 6 и 7, — важно только, чтобы в итоге осталась лишь одна запись для названия `SEA SHELLS`.

РЕШЕНИЕ

Выбираем произвольный номер ID записи, которая будет в таблице оставлена, посредством подзапроса с агрегатной функцией — например, `MIN` (в этом случае останется только запись с наименьшим значением ID):

```
1 delete from dupes
2 where id not in ( select min(id)
3                   from dupes
4                   group by name )
```

Для MySQL нужно немного подкорректировать синтаксис решения, т. к. в ней при удалении записи к одной и той же таблице нельзя обращаться больше одного раза (на момент подготовки этой книги):

```
1 delete from dupes
2 where id not in
3   (select min(id)
4   from (select id,name from dupes) tmp
5   group by name)
```

Обсуждение

Первым делом в процессе удаления дубликатов нужно точно определить, что именно делает строки дублирующимися. В рассматриваемом примере дубликатами являются записи, содержащие одинаковые значения столбца `NAME`. Имея требуемое определение, нужно выбрать какой-либо другой столбец, по которому можно будет отличать дубликаты в наборах друг от друга, чтобы определить, какой из них оставить. Наилучшим кандидатом для такого столбца (или столбцов) является первичный ключ. В нашем примере мы использовали для этой цели столбец `ID`, т. к. все записи таблицы имеют уникальные значения `ID`.

Смысл решения состоит в группировании записей по дубликатам значений (по значениям `NAME` в нашем случае) с последующим использованием агрегатной функции для выбора из этой группы по значению ключа записи, которая оставляется в таблице. Подзапрос в решении возвращает наименьшее значение `ID` для каждой группы `NAME`, которое представляет строку, оставляемую в таблице:

```
select min(id)
   from dupes
  group by name
```

```
   MIN (ID)
-----
        2
        1
        5
        4
```

Затем команда `DELETE` удаляет из таблицы все строки со значением `ID`, отсутствующим в результирующем множестве подзапроса (в нашем случае строки с `ID` 3, 6

и 7). Чтобы получить более четкое представление о процессе удаления, выполните только сам подзапрос, включив NAME в список SELECT:

```
select name, min(id)
       from dupes
       group by name
```

NAME	MIN(ID)
-----	-----
DYNAMITE	2
NAPOLEON	1
SEA SHELLS	5
SHE SELLS	4

Подзапрос возвращает строки, которые оставляются в таблице. Удаление всех других строк обеспечивает предикат NOT IN в выражении DELETE.

4.17. Удаление записей, на которые есть ссылки из другой таблицы

ЗАДАЧА

Из таблицы требуется удалить записи, на которые есть ссылки в какой-либо другой таблице. Для примера рассмотрим таблицу DEPT_ACCIDENTS, содержащую по одной строке для каждого производственного несчастного случая. Каждая строка содержит информацию об отделе, в котором произошел несчастный случай, и тип несчастного случая.

```
create table dept_accidents
( deptno integer,
  accident_name varchar(20) )
```

```
insert into dept_accidents values (10,'BROKEN FOOT')
insert into dept_accidents values (10,'FLESH WOUND')
insert into dept_accidents values (20,'FIRE')
insert into dept_accidents values (20,'FIRE')
insert into dept_accidents values (20,'FLOOD')
insert into dept_accidents values (30,'BRUISED GLUTE')
```

```
select * from dept_accidents
```

DEPTNO	ACCIDENT_NAME
-----	-----
10	BROKEN FOOT
10	FLESH WOUND
20	FIRE
20	FIRE
20	FLOOD
30	BRUISED GLUTE

Нам нужно удалить из таблицы EMP записи для служащих, работающих в отделах, в которых произошло три или больше несчастных случая.

РЕШЕНИЕ

С помощью подзапроса и агрегатной функции COUNT находим отделы с числом несчастных случаев три и больше. Затем удаляем записи для всех служащих этих отделов:

```
1 delete from emp
2   where deptno in ( select deptno
3                     from dept_accidents
4                     group by deptno
5                     having count(*) >= 3 )
```

Обсуждение

Подзапрос определяет отделы с количеством несчастных случаев три и больше:

```
select deptno
   from dept_accidents
  group by deptno
 having count(*) >= 3
```

```
DEPTNO
-----
      20
```

Затем выражение DELETE удаляет все записи служащих в возвращенных подзапросом отделах (в нашем случае только в отделе 20).

4.18. Подведем итоги

Может показаться, что вставка и обновление данных требуют меньше времени, чем запросы данных, и в последующем материале книги мы будем концентрироваться на изучении работы именно с запросами. Однако умение поддерживать данные в базе данных имеет фундаментальное значение, и приведенные здесь рецепты являются важной частью набора навыков, необходимых для работы с базами данных. Некоторые из рассмотренных команд, особенно команды для удаления записей, могут иметь необратимые последствия. Поэтому всегда предварительно просматривайте данные, предназначенные для удаления, чтобы убедиться в том, что действительно удаляются те данные, которые бесспорно подлежат удалению. Кроме этого, изучите документацию для используемой СУБД, чтобы разобраться, какие операции в ней можно отменить, а какие нельзя.

Запросы на получение метаданных

Рассматриваемые в этой главе рецепты позволят вам получить информацию о конкретной схеме базы данных. Например, о созданных вами таблицах или о неиндексированных внешних ключах. Все рассматриваемые здесь СУБД предоставляют таблицы и представления для получения таких данных. Рецепты, включенные в эту главу, помогут вам понять, как можно извлекать информацию из этих таблиц и представлений.

Хотя для хранения метаданных в таблицах и представлениях СУБД используется общая высокоуровневая стратегия, конечная ее реализация не стандартизована в той же степени, как большинство рассматриваемых в этой книге возможностей языка SQL. Поэтому, по сравнению с другими главами, в этой главе намного чаще предлагаются разные решения для разных СУБД.

Рецепты этой главы демонстрируют наиболее распространенные запросы для получения метаданных по каждой СУБД, рассматриваемой в книге. Однако информация по этой теме намного более обширна, чем представленные здесь рецепты могут охватить. Чтобы получить полный список таблиц и/или представлений каталога или словарей данных, обратитесь к документации по используемой СУБД.



Чтобы упростить демонстрацию, для всех рецептов этой главы предполагается использование схемы под именем SMEAGOL.

5.1. Создание списка таблиц схемы

ЗАДАЧА

Требуется получить список всех таблиц, созданных в той или иной схеме.

РЕШЕНИЕ

Во всех последующих решениях предполагается работа со схемой SMEAGOL. Базовый подход к решению этой задачи одинаков для всех СУБД: делается запрос по системной таблице (или представлению), содержащей в отдельной строке информацию о каждой таблице базы данных.

DB2

Делаем запрос по таблице SYSCAT.TABLES:

```

1 select tablename
2   from syscat.tables
3  where tabschema = 'SMEAGOL'

```

Oracle

Делаем запрос по таблице SYS.ALL_TABLES:

```

1 select table_name
2   from all_tables
3  where owner = 'SMEAGOL'

```

PostgreSQL, MySQL и SQL Server

Делаем запрос по таблице INFORMATION_SCHEMA.TABLES:

```

1 select table_name
2   from information_schema.tables
3  where table_schema = 'SMEAGOL'

```

Обсуждение

Базы данных предоставляют информацию о своей структуре посредством таких же механизмов, которые применяются для создания пользовательских приложений — таблиц и представлений. Например, СУБД Oracle содержит обширный каталог системных представлений ALL_TABLES, по которым можно делать запросы на получение информации о таблицах, индексах, правах доступа и любом другом объекте этой базы данных.



Представления каталога Oracle основаны на базовом наборе таблиц, содержащих информацию в очень неудобном для пользователя формате. Представления придают данным каталога Oracle более понятный формат.

Системные представления Oracle и системные таблицы DB2 по-разному реализуются разными поставщиками. С другой стороны, в PostgreSQL, MySQL и SQL Server используется так называемая *информационная схема*, состоящая из набора представлений, определенных стандартом ISO для SQL. Благодаря такому подходу для всех этих СУБД может использоваться один и тот же запрос.

5.2. Создание списка столбцов таблицы

ЗАДАЧА

Требуется создать список столбцов определенной таблицы, включающий тип данных столбцов и их позицию в таблице.

РЕШЕНИЕ

В следующих решениях предполагается использование таблицы EMP в схеме SMEAGOL.

DB2

Делаем запрос по таблице SYSCAT.COLUMNS:

```
1 select colname, typename, colno
2   from syscat.columns
3  where tabname = 'EMP'
4     and tabschema = 'SMEAGOL'
```

Oracle

Делаем запрос по таблице ALL_TAB_COLUMNS:

```
1 select column_name, data_type, column_id
2   from all_tab_columns
3  where owner = 'SMEAGOL'
4     and table_name = 'EMP'
```

PostgreSQL, MySQL и SQL Server

Делаем запрос по таблице INFORMATION_SCHEMA.COLUMNS:

```
1 select column_name, data_type, ordinal_position
2   from information_schema.columns
3  where table_schema = 'SMEAGOL'
4     and table_name = 'EMP'
```

Обсуждение

Все СУБД предоставляют средства для получения подробной информации о столбцах таблиц. Приведенные примеры запросов возвращают только названия столбцов, их тип данных и позиции в таблице. Но доступна и такая информация, как длина значений столбца, возможность сохранять в нем значения NULL и значение по умолчанию.

5.3. Создание списка индексированных столбцов таблицы

ЗАДАЧА

Требуется создать список индексов, соответствующих столбцов и позиций столбцов (при доступности такой информации) в индексе заданной таблицы.

РЕШЕНИЕ

Во всех последующих решениях для конкретных СУБД предполагается работа с таблицей EMP схемы SMEAGOL.

DB2

Делаем запрос по таблице SYSCAT.INDEXES:

```
1 select a.tabname, b.indname, b.colname, b.colseq
2   from syscat.indexes a,
3        syscat.indexcoluse b
4  where a.tabname = 'EMP'
5        and a.tabschema = 'SMEAGOL'
6        and a.indschema = b.indschema
7        and a.indname = b.indname
```

Oracle

Делаем запрос по таблице SYS.ALL_IND_COLUMNS:

```
select table_name, index_name, column_name, column_position
   from sys.all_ind_columns
  where table_name = 'EMP'
        and table_owner = 'SMEAGOL'
```

PostgreSQL

Делаем запрос по таблицам PG_CATALOG.PG_INDEXES и INFORMATION_SCHEMA.COLUMNS:

```
1 select a.tablename, a.indexname, b.column_name
2   from pg_catalog.pg_indexes a,
3        information_schema.columns b
4  where a.schemaname = 'SMEAGOL'
5        and a.tablename = b.table_name
```

MySQL

Исполняем команду SHOW INDEX:

```
show index from emp
```

SQL Server

Делаем запрос по таблицам SYS.TABLES, SYS.INDEXES, SYS.INDEX_COLUMNS и SYS.COLUMNS:

```
1 select a.name table_name,
2        b.name index_name,
3        d.name column_name,
4        c.index_column_id
```

```
5 from sys.tables a,  
6     sys.indexes b,  
7     sys.index_columns c,  
8     sys.columns d  
9 where a.object_id = b.object_id  
10 and b.object_id = c.object_id  
11 and b.index_id = c.index_id  
12 and c.object_id = d.object_id  
13 and c.column_id = d.column_id  
14 and a.name = 'EMP'
```

Обсуждение

При обращении с запросом к таблице важно знать, какие из ее столбцов проиндексированы, а какие — нет. Индексы могут повысить эффективность запросов по столбцам, которые часто используются для фильтров и которые содержат довольно большое количество однозначных значений. Индексы также полезны при объединении таблиц. Зная проиндексированные столбцы, можно избежать возможных проблем с производительностью. Кроме этого, может потребоваться информация о самих индексах: количество уровней, количество уникальных ключей, степень разветвленности и т. п. Такую информацию также можно извлечь, выполнив соответствующий запрос по таблицам и/или представлениям, рассматриваемым в решениях этого рецепта.

5.4. Создание списка ограничений таблицы

ЗАДАЧА

Требуется получить список ограничений, заданных для таблицы в определенной схеме, и столбцов, на которые они наложены. Например, надо узнать ограничения для таблицы EMP и столбцы, к которым они применяются.

РЕШЕНИЕ

DB2

Делаем запрос по таблицам SYSCAT.TABCONST и SYSCAT.COLUMNS:

```
1 select a.tabname, a.constname, b.colname, a.type  
2     from syscat.tabconst a,  
3         syscat.columns b  
4 where a.tabname = 'EMP'  
5     and a.tabschema = 'SMEAGOL'  
6     and a.tabname = b.tabname  
7     and a.tabschema = b.tabschema
```

Oracle

Делаем запрос по таблицам SYS.ALL_CONSTRAINTS и SYS.ALL_CONS_COLUMNS:

```

1 select a.table_name,
2       a.constraint_name,
3       b.column_name,
4       a.constraint_type
5 from all_constraints a,
6      all_cons_columns b
7 where a.table_name = 'EMP'
8 and a.owner = 'SMEAGOL'
9 and a.table_name = b.table_name
10 and a.owner = b.owner
11 and a.constraint_name = b.constraint_name

```

PostgreSQL, MySQL и SQL Server

Делаем запрос по таблицам INFORMATION_SCHEMA.TABLE_CONSTRAINTS и INFORMATION_SCHEMA.KEY_COLUMN_USAGE:

```

1 select a.table_name,
2       a.constraint_name,
3       b.column_name,
4       a.constraint_type
5 from information_schema.table_constraints a,
6      information_schema.key_column_usage b
7 where a.table_name = 'EMP'
8 and a.table_schema = 'SMEAGOL'
9 and a.table_name = b.table_name
10 and a.table_schema = b.table_schema
11 and a.constraint_name = b.constraint_name

```

Обсуждение

Ограничения являются настолько критичной частью реляционных баз данных, что весьма очевидна необходимость знать ограничения, наложенные на используемые таблицы. Информация об ограничениях может потребоваться по многим причинам: найти таблицы без первичного ключа, найти столбцы, которые должны быть внешними ключами, но не являются таковыми (т. е. данные дочерних таблиц отличаются от данных родительских, и нужно выяснить причину этого), или узнать проверочные ограничения. Например, допускаются ли значения NULL в столбцах? Должны ли значения столбцов удовлетворять определенному условию? И так далее.

5.5. Создание списка внешних ключей без соответствующих индексов

ЗАДАЧА

Требуется создать список таблиц, содержащих непроиндексированные столбцы внешних ключей. Например, нужно узнать, проиндексированы ли внешние ключи таблицы EMP.

РЕШЕНИЕ

DB2

Делаем запрос по таблицам SYSCAT.TABCONST, SYSCAT.KEYCOLUSE, SYSCAT.INDEXES и SYSCAT.INDEXCOLUSE:

```
1 select fkeys.tabname,
2        fkeys.constname,
3        fkeys.colname,
4        ind_cols.indname
5   from (
6 select a.tabschema, a.tabname, a.constname, b.colname
7   from syscat.tabconst a,
8        syscat.keycoluse b
9  where a.tabname = 'EMP'
10     and a.tabschema = 'SMEAGOL'
11     and a.type = 'F'
12     and a.tabname = b.tabname
13     and a.tabschema = b.tabschema
14        ) fkeys
15   left join
16   (
17 select a.tabschema,
18        a.tabname,
19        a.indname,
20        b.colname
21   from syscat.indexes a,
22        syscat.indexcoluse b
23  where a.indschema = b.indschema
24     and a.indname = b.indname
25        ) ind_cols
26   on (fkeys.tabschema = ind_cols.tabschema
27     and fkeys.tabname = ind_cols.tabname
28     and fkeys.colname = ind_cols.colname )
29  where ind_cols.indname is null
```

Oracle

Делаем запрос по таблицам SYS.ALL_CONS_COLUMNS, SYS.ALL_CONSTRAINTS и SYS.ALL_IND_COLUMNS:

```

1 select a.table_name,
2     a.constraint_name,
3     a.column_name,
4     c.index_name
5   from all_cons_columns a,
6        all_constraints b,
7        all_ind_columns c
8  where a.table_name = 'EMP'
9        and a.owner = 'SMEAGOL'
10       and b.constraint_type = 'R'
11       and a.owner = b.owner
12       and a.table_name = b.table_name
13       and a.constraint_name = b.constraint_name
14       and a.owner = c.table_owner (+)
15       and a.table_name = c.table_name (+)
16       and a.column_name = c.column_name (+)
17       and c.index_name is null

```

PostgreSQL

Делаем запрос по таблицам INFORMATION_SCHEMA.KEY_COLUMN_USAGE, INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS, INFORMATION_SCHEMA.COLUMNS и PG_CATALOG.PG_INDEXES:

```

1 select fkeys.table_name,
2     fkeys.constraint_name,
3     fkeys.column_name,
4     ind_cols.indexname
5   from (
6 select a.constraint_schema,
7     a.table_name,
8     a.constraint_name,
9     a.column_name
10  from information_schema.key_column_usage a,
11       information_schema.referential_constraints b
12 where a.constraint_name = b.constraint_name
13       and a.constraint_schema = b.constraint_schema
14       and a.constraint_schema = 'SMEAGOL'
15       and a.table_name = 'EMP'
16     ) fkeys
17  left join
18     (
19 select a.schemaname, a.tablename, a.indexname, b.column_name
20  from pg_catalog.pg_indexes a,
21       information_schema.columns b

```



```

22 where a.tablename = b.table_name
23     and a.schemaname = b.table_schema
24     ) ind_cols
25     on ( fkeys.constraint_schema = ind_cols.schemaname
26         and fkeys.table_name = ind_cols.tablename
27         and fkeys.column_name = ind_cols.column_name )
28 where ind_cols.indexname is null

```

MySQL

Получить информацию об индексах — например, название индекса, столбцы индекса, порядковая позиция столбцов индекса, можно с помощью команды `SHOW INDEX`. Кроме этого, список внешних ключей для конкретной таблицы можно получить посредством запроса по таблице `INFORMATION_SCHEMA.KEY_COLUMN_USAGE`. Можно было бы предположить, что в MySQL внешние ключи индексируются автоматически, но в действительности иногда этого не происходит. Определить, был ли проиндексирован столбец внешнего ключа определенной таблицы, можно, выполнив для нее команду `SHOW INDEX`, а затем сравнив полученный результат с содержимым таблицы `INFORMATION_SCHEMA.KEY_COLUMN_USAGE`. `COLUMN_NAME` для этой же таблицы. Если `COLUMN_NAME` есть в `KEY_COLUMN_USAGE`, но отсутствует в результатах команды `SHOW INDEX`, делаем вывод, что этот столбец не проиндексирован.

SQL Server

Делаем запрос по таблицам `SYS.TABLES`, `SYS.FOREIGN_KEYS`, `SYS.COLUMNS`, `SYS.INDEXES` и `SYS.INDEX_COLUMNS`:

```

1 select fkeys.table_name,
2     fkeys.constraint_name,
3     fkeys.column_name,
4     ind_cols.index_name
5     from (
6 select a.object_id,
7     d.column_id,
8     a.name table_name,
9     b.name constraint_name,
10    d.name column_name
11    from sys.tables a
12         join
13         sys.foreign_keys b
14         on ( a.name = 'EMP'
15             and a.object_id = b.parent_object_id
16           )
17         join
18         sys.foreign_key_columns c
19         on ( b.object_id = c.constraint_object_id )
20

```

```

21      sys.columns d
22  on (      c.constraint_column_id = d.column_id
23      and a.object_id              = d.object_id
24      )
25      ) fkeys
26      left join
27      (
28  select a.name index_name,
29         b.object_id,
30         b.column_id
31  from sys.indexes a,
32         sys.index_columns b
33  where a.index_id = b.index_id
34         ) ind_cols
35  on (      fkeys.object_id = ind_cols.object_id
36         and fkeys.column_id = ind_cols.column_id )
37  where ind_cols.index_name is null

```

Обсуждение

При изменении содержимого строк в СУБД разных поставщиков используются разные блокировочные механизмы. При организации отношения типа «родитель — потомок» посредством внешнего ключа индексация дочернего столбца (столбцов) может уменьшить объем блокировок (подробную информацию по этому вопросу ищите в документации на используемую СУБД). В других случаях дочерняя таблица часто объединяется с родительской по столбцу внешнего ключа, поэтому наличие индекса может повысить производительность и в такой ситуации.

5.6. Генерирование кода SQL с помощью средств SQL

ЗАДАЧА

Требуется генерировать динамические выражения SQL — например, чтобы автоматизировать выполнение задач поддержки базы данных. В частности, нужно реализовать три задачи: подсчет количества строк в таблицах, отключение ограничений во внешних ключах, заданных в таблицах, и генерацию сценариев вставки из данных таблиц.

РЕШЕНИЕ

Идея состоит в использовании строк для создания выражений SQL, а данные для вставки (например, название объекта для обработки командой) предоставляются данными из таблиц, из которых осуществляется выборка. При этом имейте в виду, что запросы только создают такие выражения, а исполнять их нужно вручную — посредством сценария или любым другим способом. Запросы в приведенных далее примерах предназначены для исполнения на СУБД Oracle. Для других СУБД при-

меняется точно такой же метод, различаются лишь названия таблиц словаря данных и форматирование дат. В приведенных затем результатах запросов показана часть строк результирующего множества СУБД Oracle на моем ноутбуке. Ваши результаты будут, конечно же, другими:

```
/* Генерируем код SQL для подсчета всех строк во всех таблицах */
```

```
select 'select count(*) from '||table_name||';' cnts
      from user_tables;
```

CNTS

```
-----
select count(*) from ANT;
select count(*) from BONUS;
select count(*) from DEMO1;
select count(*) from DEMO2;
select count(*) from DEPT;
select count(*) from DUMMY;
select count(*) from EMP;
select count(*) from EMP_SALES;
select count(*) from EMP_SCORE;
select count(*) from PROFESSOR;
select count(*) from T;
select count(*) from T1;
select count(*) from T2;
select count(*) from T3;
select count(*) from TEACH;
select count(*) from TEST;
select count(*) from TRX_LOG;
select count(*) from X;
```

```
/* Отключаем внешние ключи для всех таблиц */
```

```
select 'alter table '||table_name||
      ' disable constraint '||constraint_name||';' cons
      from user_constraints
      where constraint_type = 'R';
```

CONS

```
-----
alter table ANT disable constraint ANT_FK;
alter table BONUS disable constraint BONUS_FK;
alter table DEMO1 disable constraint DEMO1_FK;
alter table DEMO2 disable constraint DEMO2_FK;
alter table DEPT disable constraint DEPT_FK;
alter table DUMMY disable constraint DUMMY_FK;
alter table EMP disable constraint EMP_FK;
alter table EMP_SALES disable constraint EMP_SALES_FK;
```

```
alter table EMP_SCORE disable constraint EMP_SCORE_FK;
alter table PROFESSOR disable constraint PROFESSOR_FK;
```

/* Генерируем сценарий для вставки данных из некоторых столбцов таблицы EMP */

```
select 'insert into emp(empno,ename,hiredate) '||chr(10)||
       'values( '||empno||','||' '||ename
       ||','||to_date('||' '||hiredate||' '||') );' inserts
       from emp
       where deptno = 10;
```

INSERTS

```
-----
insert into emp(empno,ename,hiredate)
values( 7782,'CLARK',to_date('09-JUN-2006 00:00:00') );
insert into emp(empno,ename,hiredate)
values( 7839,'KING',to_date('17-NOV-2006 00:00:00') );
insert into emp(empno,ename,hiredate)
values( 7934,'MILLER',to_date('23-JAN-2007 00:00:00') );
```

Обсуждение

Использование средств SQL для генерирования кода SQL особенно удобно при создании переносимых сценариев — например, для тестирования в нескольких средах. Кроме этого, как можно видеть в приведенных примерах, этот метод также можно задействовать для выполнения групповых операций по обслуживанию базы данных и для получения информации сразу о нескольких объектах. Генерирование кода SQL с помощью средств SQL — чрезвычайно простая операция, и чем больше экспериментировать с ней, тем понятнее она становится. Приведенные примеры должны составить хорошую основу для создания своих «динамических» сценариев SQL, потому что, откровенно говоря, в этом нет ничего сложного. Просто работайте над этим, и у вас все получится.

5.7. Описание представлений словаря данных в базе данных Oracle

ЗАДАЧА

Вы не можете вспомнить, какие представления словаря данных вам доступны, а также не можете вспомнить определения их столбцов. Что еще хуже, у вас нет под рукой документации на эту СУБД.

РЕШЕНИЕ

Этот рецепт применим только для Oracle. СУБД Oracle не только содержит солидный набор представлений словарей данных, но и представления словарей данных,

документирующие представления словарей данных. Такая себе замечательная круговая поддержка.

Чтобы создать список представлений словарей данных и их назначение, выполняем запрос по представлению DICTONARY:

```
select table_name, comments
       from dictionary
       order by table_name;
```

TABLE_NAME	COMMENTS
ALL_ALL_TABLES	Description of all object and relational tables accessible to the user
ALL_APPLY	Details about each apply process that dequeues from the queue visible to the current user

Чтобы получить описание столбцов этого представления словаря данных, выполняем запрос по представлению DICT_COLUMNS:

```
select column_name, comments
       from dict_columns
       where table_name = 'ALL_TAB_COLUMNS';
```

COLUMN_NAME	COMMENTS
OWNER	
TABLE_NAME	Table, view or cluster name
COLUMN_NAME	Column name
DATA_TYPE	Datatype of the column
DATA_TYPE_MOD	Datatype modifier of the column
DATA_TYPE_OWNER	Owner of the datatype of the column
DATA_LENGTH	Length of the column in bytes
DATA_PRECISION	Length: decimal digits (NUMBER) or binary digits (FLOAT)

Обсуждение

Раньше, когда набор документации Oracle не имелся в таком свободном доступе в Сети, наличие в этой СУБД представлений DICTONARY и DICT_COLUMNS было крайне удобным. На основе информации только из этих двух представлений можно было постепенно получить информацию обо всех других представлениях, использование которой уже давало возможность перейти к изучению всей базы данных.

Информация из представлений DICTONARY и DICT_COLUMNS полезна даже и в настоящее время. Часто, если неизвестно, какое представление описывает тот или иной тип объектов, эту информацию можно получить, выполнив групповой запрос. Например, следующий запрос поможет понять, какие представления могут описывать таблицы используемой схемы:

```
select table_name, comments
       from dictionary
       where table_name LIKE '%TABLE%'
       order by table_name;
```

Он возвращает названия всех представлений словарей данных, содержащих термин TABLE. Этот подход основан на достаточно последовательных правилах именования представлений словарей данных СУБД Oracle. Названия всех представлений, описывающих таблицы, с большой вероятностью содержат слово TABLE. Но иногда, как в случае с представлением ALL_TAB_COLUMNS, слово TABLE сокращается до просто TAB.

5.8. Подведем итоги

Запросы по метаданным позволяют переложить большую часть работы с плеч разработчика на SQL, а также освобождают его от необходимости знать некоторые детали используемой базы данных. Они особенно полезны при работе с более сложными базами данных с соответственно более сложными структурами.

Работа со строками

В этой главе мы рассмотрим, как использовать средства SQL для работы со строками. При этом следует иметь в виду, что язык SQL не предназначен для сложных манипуляций со строками, и такие операции иногда могут оказаться (и, скорее всего, окажутся) неуклюжими и утомительными. Но, несмотря на такую ограниченность, разные СУБД предоставляют определенные встроенные функции, которые мы попытались творчески использовать. Материал этой главы особенно хорошо отражает суть послания, которое мы пытались передать во введении: SQL одновременно может быть хорошим, плохим и ужасным. Мы надеемся, что эта глава позволит вам лучше оценить, что можно и что нельзя делать со строками в SQL. Во многих случаях парсинг и преобразование строк даются на удивление легко, тогда как в других код SQL, необходимый для выполнения той или иной задачи, просто приводит в ужас.

Функции `TRANSLATE` и `REPLACE`, которые используются во многих из предлагаемых здесь рецептов, в настоящее время поддерживаются практически всеми рассматриваемыми в книге СУБД. Исключение составляет MySQL, которая поддерживает только `REPLACE`. Но и в этом случае эффект `TRANSLATE` можно легко воспроизвести, используя вложенные функции `REPLACE`.

Важность первого рецепта главы трудно переоценить, т. к. он используется в целом ряде последующих решений. И действительно, во многих случаях желательно иметь возможность посимвольного обхода строки. К сожалению, в SQL нет средств для легкого решения этой задачи. Вследствие ограниченной поддержки циклов в SQL, для обхода строк цикл нужно имитировать. Мы называем эту операцию «проходом строки» или «проходом по строке» и рассматриваем ее в самом первом рецепте. Это фундаментальная операции для парсинга строк в SQL, которая используется или упоминается почти во всех рецептах этой главы. Мы настоятельно рекомендуем хорошо разобраться в работе этого метода.

6.1. Проход строки

ЗАДАЧА

Требуется выполнить проход по строке, чтобы вернуть каждый ее символ в отдельной строке таблицы, хотя в SQL нет операции цикла. Например, надо отобразить значение `KING` столбца `ENAME` таблицы `EMP` в четырех строках, каждая из которых содержит по одному символу значения.

РЕШЕНИЕ

Используем декартово произведение, чтобы получить количество строк, необходимых для возвращения каждого символа строки значения в отдельной строке. Затем с помощью встроенной функции парсинга используемой СУБД извлекаем требуемые символы (для SQL Server замените SUBSTR на SUBSTRING, а LENGTH — на DATALENGTH).

```
1 select substr(e.ename,iter.pos,1) as C
2   from (select ename from emp where ename = 'KING') e,
3        (select id as pos from t10) iter
4  where iter.pos <= length(e.ename)
```

```
C
-
K
I
N
G
```

Обсуждение

Ключевой момент посимвольной обработки строки — выполнение объединения с таблицей, имеющей достаточное количество строк для создания требуемого числа итераций. Для этого в приведенном примере используется таблица T10, содержащая 10 строк, и только один столбец ID, в котором хранятся значения от 1 до 10. Следовательно, этот запрос может вернуть максимум 10 строк.

В следующем примере показано декартово произведение представлений E и ITER (т. е. конкретного имени и 10 строк таблицы T10) без выполнения парсинга значения столбца ENAME:

```
select ename, iter.pos
   from (select ename from emp where ename = 'KING') e,
        (select id as pos from t10) iter
```

ENAME	POS
KING	1
KING	2
KING	3
KING	4
KING	5
KING	6
KING	7
KING	8
KING	9
KING	10

Кардинальность вложенного представления E равна 1, а вложенного представления ITER — 10. Таким образом, декартово произведение этих двух представлений будет

равно 10 строкам. Первый шаг имитирования цикла в SQL и состоит в создании такого произведения.



Таблицы типа T10 обычно называются «сводными».

В приведенном решении для выхода из цикла после возвращения четырех строк используется предикат `WHERE`. Для ограничения количества строк результирующего множества количеством символов в имени в этом предикате накладывается условие `ITER.POS <= LENGTH(E.ENAME)`:

```
select ename, iter.pos
  from (select ename from emp where ename = 'KING') e,
       (select id as pos from t10) iter
 where iter.pos <= length(e.ename)
```

ENAME	POS
KING	1
KING	2
KING	3
KING	4

Теперь, когда каждый символ `E.ENAME` находится в отдельной строке таблицы, по-символьный проход строки можно выполнить, передав функции `SUBSTR` в качестве параметра значение `ITER.POS`. Это значение инкрементируется для каждой строки, позволяя, таким образом, возвращать очередной символ `E.ENAME`. Вот так и работает этот пример решения.

В зависимости от поставленной задачи, не обязательно выводить каждый символ исходной строки в отдельную строку результата. В следующем примере запроса выполняется проход строки `E.NAME` с выводом в отдельную строку результата разных частей строки (более одного символа):

```
select substr(e.ename,iter.pos) a,
       substr(e.ename,length(e.ename)-iter.pos+1) b
  from (select ename from emp where ename = 'KING') e,
       (select id pos from t10) iter
 where iter.pos <= length(e.ename)
```

A	B
KING	G
ING	NG
NG	ING
G	KING

Рецепты этой главы наиболее часто используются в таких сценариях, как проход всей строки с выводом каждого ее символа в отдельной строке результата или про-

ход строки таким образом, чтобы создать количество строк, равное количеству определенных символов или разделителей исходной строки.

6.2. Вставка кавычек в строковые константы

ЗАДАЧА

Требуется вставлять символ кавычек в строковые константы. Например, нужно получить результат наподобие следующего:

```
QMARKS
-----
g'day mate
beavers' teeth
```

РЕШЕНИЕ

Следующие выражения `SELECT` демонстрируют три разных способа вывода кавычек в результатах запроса — как посредине строки, так и по отдельности:

```
1 select 'g'day mate' qmarks from t1 union all
2 select 'beavers'' teeth'   from t1 union all
3 select ''''                from t1
```

Обсуждение

При работе с кавычками часто будет удобно рассматривать их как скобки. В случае скобок для каждой открывающей скобки должна быть закрывающая. То же относится и к кавычкам. Имейте в виду, что количество кавычек в любой строке должно быть четным. Чтобы вставить в строку одну кавычку, в запросе нужно указать две:

```
select 'apples core', 'apple''s core',
       case when '' is null then 0 else 1 end
from t1

'APPLESCORE 'APPLE''SCOR CASEWHEN''ISNULLTHEN0ELSE1END
-----
apples core apple's core
```

Далее приводится решение, сведенное к базовым элементам. Две внешние кавычки определяют строковую константу, в которой две кавычки представляют одну, которая в действительности отображается в результате:

```
select '''' as quote from t1
```

```
Q
-
```

При работе с кавычками следует помнить, что строковая константа из двух кавычек, не содержащая ничего между ними, представляет значение `NULL`.

6.3. Подсчет количества вхождений в строку определенного символа

ЗАДАЧА

Требуется подсчитать количество вхождений символа или подстроки в заданную строку. Возьмем, например, следующую строку:

```
10, CLARK, MANAGER
```

Надо определить в ней количество символов запятой.

РЕШЕНИЕ

Чтобы определить количество запятых в строке, вычитаем из общего количества символов строки количество символов в ней без запятых. Все СУБД оснащены функциями для определения длины строки и удаления символов из строки. В большинстве случаев это функции `LENGTH` и `REPLACE`, соответственно (в SQL Server вместо `LENGTH` используется встроенная функция `LEN`).

```
1 select (length('10,CLARK,MANAGER')-
2       length(replace('10,CLARK,MANAGER',' ','')))/length(',')
3       as cnt
4 from t1
```

Обсуждение

Эта задача решается простой операцией вычитания. Вызываемая в строке 1 функция `LENGTH` возвращает длину исходной строки, а первый вызов этой функции в строке 2 возвращает длину строки без запятых, которые удаляются функцией `REPLACE`.

Разность при вычитании второго значения из первого и будет количеством запятых в исходной строке. В последней операции эта разность делится на длину искомой строки. Операция деления необходима, если искомая подстрока содержит более одного символа. В следующем примере подсчет вхождений символов `LL` в строку `HELLO HELLO` без деления возвратит неправильный результат:

```
select
  (length('HELLO HELLO')-
  length(replace('HELLO HELLO','LL','')))/length('LL')
  as correct_cnt,
  (length('HELLO HELLO')-
  length(replace('HELLO HELLO','LL',''))) as incorrect_cnt
from t1
```

```
CORRECT_CNT INCORRECT_CNT
```

6.4. Удаление символов из строки

ЗАДАЧА

Требуется удалить из данных определенные символы. Например, в финансовых данных надо удалить запятые, разделяющие группы цифр, или символы валют, сопутствующие значениям количества. Другой пример: нужно перенести данные из базы данных в файл CSV, но данные содержат текстовое поле с запятыми, которые рассматриваются в формате CSV в качестве разделителей. Или же рассмотрим следующее результирующее множество:

ENAME	SAL
SMITH	800
ALLEN	1600
WARD	1250
JONES	2975
MARTIN	1250
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
TURNER	1500
ADAMS	1100
JAMES	950
FORD	3000
MILLER	1300

Из этих данных нам нужно убрать все гласные буквы и нули, чтобы получить следующий результат (соответственно столбцы STRIPPED1 и STRIPPED2):

ENAME	STRIPPED1	SAL	STRIPPED2
SMITH	SMTH	800	8
ALLEN	LLN	1600	16
WARD	WRD	1250	125
JONES	JNS	2975	2975
MARTIN	MRTN	1250	125
BLAKE	BLK	2850	285
CLARK	CLRK	2450	245
SCOTT	SCTT	3000	3
KING	KNG	5000	5
TURNER	TRNR	1500	15
ADAMS	DMS	1100	11
JAMES	JMS	950	95
FORD	FRD	3000	3
MILLER	MLLR	1300	13

РЕШЕНИЕ

Все СУБД поддерживают функции для удаления символов из строки. Для решения этой задачи лучше всего подойдут функции `REPLACE` и `TRANSLATE`.

DB2, Oracle, PostgreSQL и SQL Server

Для удаления заданных символов используем встроенные функции `TRANSLATE` и `REPLACE`:

```
1 select ename,
2         replace(translate(ename,'aaaaa','AEIOU'),'a','') as stripped1,
3         sal,
4         replace(cast(sal as char(4)),'0','') as stripped2
5   from emp
```

Обратите внимание, что для DB2 ключевое слово `AS` нужно только для присвоения псевдонима столбцу и использовать его не обязательно.

MySQL

Отсутствующая в MySQL функция `TRANSLATE` заменяется здесь несколькими вызовами функции `REPLACE`:

```
1 select ename,
2         replace(
3         replace(
4         replace(
5         replace(
6         replace(ename,'A',''),'E',''),'I',''),'O',''),'U','')
7         as stripped1,
8         sal,
9         replace(sal,0,'') stripped2
10  from emp
```

Обсуждение

В решении все вхождения символа 0 удаляются с помощью функции `REPLACE`. Гласные буквы помогает удалить функция `TRANSLATE`, которая преобразовывает их в какой-либо один символ (в приведенном примере это символ `a`, но можно взять и любой другой символ), который затем удаляется посредством уже знакомой функции `REPLACE`.

6.5. Разделение цифровых и символьных данных

ЗАДАЧА

У нас есть числовые данные, перемешанные с символьными в одном столбце. Такая ситуация может возникнуть, например, с унаследованными данными, в которых

вместе со значениями количества или стоимости товара были также сохранены соответствующие единицы измерения или обозначения денежных единиц (вместо обозначения столбца соответствующей единицей измерения или использования для данных отдельного столбца в них сохранены значения типа «100 км», или «AUD\$200» или «40 фунтов»).

Итак, нам нужно отделить символьные данные от числовых. Рассмотрим, например, следующее результирующее множество с одним столбцом:

```
DATA
-----
SMITH800
ALLEN1600
WARD1250
JONES2975
MARTIN1250
BLAKE2850
CLARK2450
SCOTT3000
KING5000
TURNER1500
ADAMS1100
JAMES950
FORD3000
MILLER1300
```

Мы хотим получить из него следующий результат с двумя столбцами:

ENAME	SAL
SMITH	800
ALLEN	1600
WARD	1250
JONES	2975
MARTIN	1250
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
TURNER	1500
ADAMS	1100
JAMES	950
FORD	3000
MILLER	1300

РЕШЕНИЕ

Для отделения символьных данных от числовых используем встроенные функции `TRANSLATE` и `REPLACE`. Подобно другим рецептам этой книги, хитрость состоит в использовании функции `TRANSLATE` для преобразования нескольких символов в один,

к которому можно обращаться. Тогда нам больше не понадобится искать разные числа или символы, а только один символ, представляющий все числа или все символьные строки.

DB2

Для разделения символьных и числовых данных используем встроенные функции TRANSLATE и REPLACE:

```

1 select replace(
2     translate(data, '0000000000', '0123456789'), '0', '') ename,
3     cast(
4     replace(
5     translate(lower(data), repeat('z', 26),
6         'abcdefghijklmnopqrstuvwxyz'), 'z', '') as integer) sal
7 from (
8 select ename||cast(sal as char(4)) data
9 from emp
10      ) x

```

Oracle

Для разделения символьных и числовых данных используем встроенные функции TRANSLATE и REPLACE:

```

1 select replace(
2     translate(data, '0123456789', '0000000000'), '0') ename,
3     to_number(
4     replace(
5     translate(lower(data),
6         'abcdefghijklmnopqrstuvwxyz',
7         rpad('z', 26, 'z')), 'z')) sal
8 from (
9 select ename||sal data
10 from emp
11      )

```

PostgreSQL

Для разделения символьных и числовых данных используем встроенные функции TRANSLATE и REPLACE:

```

1 select replace(
2     translate(data, '0123456789', '0000000000'), '0', '') as ename,
3     cast(
4     replace(
5     translate(lower(data),
6         'abcdefghijklmnopqrstuvwxyz',
7         rpad('z', 26, 'z')), 'z', '') as integer) as sal

```

```

8  from (
9  select ename||sal as data
10 from emp
11      ) x

```

SQL Server

Для разделения символьных и числовых данных используем встроенные функции TRANSLATE и REPLACE:

```

1  select replace(
2      translate(data, '0123456789', '0000000000'), '0', '') as ename,
3      cast(
4      replace(
5      translate(lower(data),
6          'abcdefghijklmnopqrstuvwxyz',
7          replicate('z',26), 'z', '') as integer) as sal
8  from (
9  select concat(ename, sal) as data
10 from emp
11      ) x

```

Обсуждение

Синтаксис решений для каждой СУБД несколько иной, но базовый принцип одинаков для всех, поэтому работу решения мы рассмотрим на примере для Oracle. Ключ к решению задачи — изолировать числовые и символьные данные друг от друга. Это решается с помощью функций TRANSLATE и REPLACE. Чтобы извлечь числовые данные, сначала выделяем символьные данные с помощью функции TRANSLATE:

```

select data,
       translate(lower(data),
                'abcdefghijklmnopqrstuvwxyz',
                rpad('z',26,'z')) sal
from (select ename||sal data from emp)

```

DATA	SAL
SMITH800	zzzzz800
ALLEN1600	zzzzz1600
WARD1250	zzzz1250
JONES2975	zzzzz2975
MARTIN1250	zzzzz1250
BLAKE2850	zzzzz2850
CLARK2450	zzzzz2450
SCOTT3000	zzzzz3000
KING5000	zzzz5000
TURNER1500	zzzzz1500
ADAMS1100	zzzzz1100


```
JAMES950          zzzzz950
FORD3000          zzzz3000
MILLER1300        zzzzzz1300
```

Здесь посредством функции `TRANSLATE` все нечисловые символы преобразуются в строчную букву `z`. На следующем этапе с помощью функции `REPLACE` удаляем из всех записей все вхождения строчной буквы `z`, оставляя только числовые символы, которые потом можно привести к числовому типу данных:

```
select data,
       to_number(
         replace(
           translate(lower(data),
                    'abcdefghijklmnopqrstuvwxyz',
                    rpad('z',26,'z')), 'z')) sal
  from (select ename||sal data from emp)
```

DATA	SAL
SMITH800	800
ALLEN1600	1600
WARD1250	1250
JONES2975	2975
MARTIN1250	1250
BLAKE2850	2850
CLARK2450	2450
SCOTT3000	3000
KING5000	5000
TURNER1500	1500
ADAMS1100	1100
JAMES950	950
FORD3000	3000
MILLER1300	1300

Чтобы извлечь нечисловые символы, сначала выделим все числовые символы с помощью функции `TRANSLATE`:

```
select data,
       translate(data, '0123456789', '0000000000') ename
  from (select ename||sal data from emp)
```

DATA	ENAME
SMITH800	SMITH000
ALLEN1600	ALLEN0000
WARD1250	WARD0000
JONES2975	JONES0000
MARTIN1250	MARTIN0000
BLAKE2850	BLAKE0000
CLARK2450	CLARK0000

SCOTT3000	SCOTT0000
KING5000	KING0000
TURNER1500	TURNER0000
ADAMS1100	ADAMS0000
JAMES950	JAMES000
FORD3000	FORD0000
MILLER1300	MILLER0000

Здесь функция `TRANSLATE` используется для преобразования всех числовых символов в символ `0`. На следующем этапе с помощью функции `REPLACE` удаляем из всех записей все вхождения символа `0`, оставляя только нечисловые символы:

```
select data,
       replace(translate(data, '0123456789', '0000000000'), '0') ename
from (select ename||sal data from emp)
```

DATA	ENAME
-----	-----
SMITH800	SMITH
ALLEN1600	ALLEN
WARD1250	WARD
JONES2975	JONES
MARTIN1250	MARTIN
BLAKE2850	BLAKE
CLARK2450	CLARK
SCOTT3000	SCOTT
KING5000	KING
TURNER1500	TURNER
ADAMS1100	ADAMS
JAMES950	JAMES
FORD3000	FORD
MILLER1300	MILLER

Совместное выполнение этих обеих операций обеспечивает решение поставленной задачи.

6.6. Определение, содержит ли строка только буквенно-цифровые символы

ЗАДАЧА

Вы хотите возвращать строки из таблицы только в том случае, когда заданный столбец не содержит никаких иных символов, кроме цифр и букв. Рассмотрим следующее представление `V` (для SQL Server вместо оператора конкатенации `||` используется оператор `+`):

```
create view V as
select ename as data
from emp
```

```

where deptno=10
union all
select ename||', $'|| cast(sal as char(4)) ||'.00' as data
  from emp
where deptno=20
union all
select ename|| cast(deptno as char(4)) as data
  from emp
where deptno=30

```

Представление V показывает нашу таблицу и возвращает следующие данные:

```

DATA
-----
CLARK
KING
MILLER
SMITH, $800.00
JONES, $2975.00
SCOTT, $3000.00
ADAMS, $1100.00
FORD, $3000.00
ALLEN30
WARD30
MARTIN30
BLAKE30
TURNER30
JAMES30

```

Но из всех этих данных представления нам нужны только следующие записи:

```

DATA
-----
CLARK
KING
MILLER
ALLEN30
WARD30
MARTIN30
BLAKE30
TURNER30
JAMES30

```

Иными словами, мы хотим вернуть лишь те строки, которые содержат только цифры и буквы.

РЕШЕНИЕ

С первого взгляда интуитивным решением этой задачи может показаться поиск всех символов строки, не являющихся буквенно-цифровыми. Однако более легким решением будет, напротив, в точности обратное: найти все буквенно-цифровые

символы. Для этого их нужно заменить каким-либо одним символом и рассматривать их все как один символ. Это позволит нам манипулировать всеми буквенно-цифровыми символами, как одним целым, — в копии строки, в которой все буквенно-цифровые символы представлены каким-либо одним символом, отделить все буквенно-цифровые символы от любых других символов не составит никакого труда.

DB2

Сначала с помощью функции `TRANSLATE` преобразовываем все буквенно-цифровые символы в какой-либо один символ, а затем определяем строки, содержащие иные символы, чем символ замены буквенно-цифровых символов. Для DB2 в представлении `V` необходимо вызывать функцию `CAST`, т. к. в противном случае создать представление не получится из-за ошибок при преобразовании типов. Выполняя приведение к типу `CHAR`, нужно быть особо внимательным, учитывая фиксированную длину этого типа (с дополнением пробелами):

```
1 select data
2   from V
3  where translate(lower(data),
4                  repeat('a',36),
5                  '0123456789abcdefghijklmnopqrstuvwxyz') =
6                  repeat('a',length(data))
```

MySQL

Для MySQL синтаксис запроса для создания представления `V` немного другой:

```
create view V as
select ename as data
  from emp
 where deptno=10
 union all
select concat(ename,'$',sal,'.00') as data
  from emp
 where deptno=20
 union all
select concat(ename,deptno) as data
  from emp
 where deptno=30
```

Для извлечения строк, содержащих не только буквенно-цифровые данные, используем регулярное выражение:

```
1 select data
2   from V
3  where data regexp '^[0-9a-zA-Z]' = 0
```

Oracle и PostgreSQL

Сначала с помощью функции `TRANSLATE` преобразовываем все буквенно-цифровые символы в какой-либо один символ. Затем определяем строки, содержащие иные символы, чем символ замены буквенно-цифровых символов. Для этих СУБД вызывать функцию `CAST` в представлении `V` не требуется. Но, так же как и для DB2, при выполнении приведения к типу `CHAR` нужно быть особо внимательным, учитывая фиксированную длину этого типа (с дополнением пробелами). Если необходимо выполнить приведение к символьному типу данных, используйте типы `VARCHAR` или `VARCHAR2`:

```
1 select data
2   from V
3  where translate(lower(data),
4                  '0123456789abcdefghijklmnopqrstuvwxy',
5                  rpad('a',36,'a')) = rpad('a',length(data),'a')
```

SQL Server

Здесь используется такой же подход, только не задействуется функция `RPAD`:

```
1 select data
2   from V
3  where translate(lower(data),
4                  '0123456789abcdefghijklmnopqrstuvwxy',
5                  replicate('a',36)) = replicate('a',len(data))
```

Обсуждение

Ключевой момент этих решений — возможность одновременного обращения к нескольким символам. Функция `TRANSLATE` позволяет с легкостью манипулировать всеми числами или символами без необходимости посимвольной проверки символов в «цикле».

DB2, Oracle, PostgreSQL и SQL Server

Из 14 строк представления `V` только 9 являются буквенно-цифровыми. Строки, содержащие только буквенно-цифровые данные, обнаруживаются с помощью функции `TRANSLATE`. В приведенном примере функция `TRANSLATE` преобразовывает все символы 0–9 и a–z в символ `a`. Преобразованная таким образом строка сравнивается со строкой из символов `a` такой же длины. Если длина строк совпадает, мы знаем, что исходная строка содержит только буквенно-цифровые символы.

Итак, посредством функции `TRANSLATE` (рассматриваем синтаксис для Oracle):

```
where translate(lower(data),
                '0123456789abcdefghijklmnopqrstuvwxy',
                rpad('a',36,'a'))
```

преобразовываем все цифры и буквы в какой-либо один символ (в нашем примере в `a`). После такого преобразования все строки, которые действительно являются буквенно-цифровыми, можно идентифицировать как строки, состоящие лишь из одного символа (в нашем случае `a`). В этом можно убедиться, выполнив только саму функцию `TRANSLATE`:

```
select data, translate(lower(data),
                       '0123456789abcdefghijklmnopqrstuvwxyz',
                       rpad('a',36,'a'))
from V
```

DATA	TRANSLATE (LOWER (DATA))
CLARK	aaaaa
...	
SMITH, \$800.00	aaaaa, \$aaa.aa
...	
ALLEN30	aaaaaaaa
...	

Все буквенно-цифровые значения преобразованы, но длина строк не изменилась. Поэтому оставляем только те строки, для которых вызов функции `TRANSLATE` возвращает строку, состоящую из одних `a`, и отбрасываем остальные. Для этого сравниваем длину каждой исходной строки с длиной соответствующей ей строки из символов `a`:

```
select data, translate(lower(data),
                       '0123456789abcdefghijklmnopqrstuvwxyz',
                       rpad('a',36,'a')) translated,
       rpad('a',length(data),'a') fixed
from V
```

DATA	TRANSLATED	FIXED
CLARK	aaaaa	aaaaa
...		
SMITH, \$800.00	aaaaa, \$aaa.aa	aaaaaaaaaaaaaaaa
...		
ALLEN30	aaaaaaaa	aaaaaaaa
...		

В заключение оставляем только те строки, для которых значение столбца `TRANSLATED` равно значению столбца `FIXED`.

MySQL

Регулярное выражение в предикате `WHERE`:

```
where data regexp '^[^0-9a-zA-Z]' = 0
```

вызывает возвращение строк, содержащих только цифры или буквы. Диапазоны значений в квадратных скобках: `0-9a-zA-Z` — представляют все возможные цифры и

буквы. Символ `^` задает отрицание, т. е. заключенное в квадратные скобки выражение означает: «ни цифры, ни буквы». Возвращаемое значение `1` означает удовлетворение условия, а значение `0` — неудовлетворение. Таким образом, все представленное выражение означает: «возвращение строк, содержащих что-либо, кроме цифр и букв, будет ошибкой».

6.7. Извлечение инициалов из имен

ЗАДАЧА

Требуется преобразовать полное имя в инициалы. Например, из этого имени:

```
Stewie Griffin
```

нужно получить:

```
S.G.
```

РЕШЕНИЕ

Важно иметь в виду, что SQL не обладает гибкостью таких языков, как C или Python, поэтому в нем не так-то просто создать общее решение для обработки любых форматов имени. В рассматриваемых далее решениях ожидается, что имена будут в формате «Имя Фамилия» или «Имя Отчество/Инициал отчества Фамилия».

DB2

Для извлечения инициалов используем встроенные функции `REPLACE`, `TRANSLATE` и `REPEAT`:

```
1 select replace(
2     replace(
3         translate(replace('Stewie Griffin', '.', ''),
4                 repeat('#',26),
5                 'abcdefghijklmnopqrstuvwxyz'),
6         '#',''), ' ','.')
7     || '.'
8 from t1
```

MySQL

Для извлечения инициалов используем встроенные функции `CONCAT`, `CONCAT_WS`, `SUBSTRING` и `SUBSTRING_INDEX`:

```
1 select case
2     when cnt = 2 then
3         trim(trailing '.' from
4             concat_ws('.',
5                 substr(substring_index(name, ' ',1),1,1),
```

```

6         substr(name,
7             length(substring_index(name, ' ',1))+2,1),
8         substr(substring_index(name, ' ',-1),1,1),
9         '.'))
10    else
11        trim(trailing '.' from
12            concat_ws('.',
13                substr(substring_index(name, ' ',1),1,1),
14                substr(substring_index(name, ' ',-1),1,1)
15            ))
16    end as initials
17  from (
18  select name,length(name)-length(replace(name, ' ', '')) as cnt
19  from (
20  select replace('Stewie Griffin','.', '') as name from t1
21         )y
22         )x

```

Oracle и PostgreSQL

Для извлечения инициалов используем встроенные функции REPLACE, TRANSLATE и RPAD:

```

1  select replace(
2     replace(
3     translate(replace('Stewie Griffin', '.', ''),
4               'abcdefghijklmnopqrstuvwxyz',
5               rpad('#',26,'#') ), '#', '' ), ' ', '.' ) || '.'
6  from t1

```

SQL Server

```

1  select replace(
2     replace(
3     translate(replace('Stewie Griffin', '.', ''),
4               'abcdefghijklmnopqrstuvwxyz',
5               replicate('#',26) ), '#', '' ), ' ', '.' ) + '.'
6  from t1

```

Обсуждение

Инициалы из имени/фамилии можно извлечь, выделив в них заглавные буквы. Далее приводятся специфичные для каждой рассматриваемой здесь СУБД подробности решений.

DB2

Функция REPLACE удаляет из имени все символы точки (для обработки второго инициала), а функция TRANSLATE преобразовывает все строчные буквы в символ #:


```
select translate(replace('Stewie Griffin', '.', ''),
                repeat('#',26),
                'abcdefghijklmnopqrstuvwxyz')
from t1
```

```
TRANSLATE('STE
-----
S##### G#####
```

В полученном результате инициалами являются все символы, отличные от #. Далее функция REPLACE удаляет все символы #:

```
select replace(
    translate(replace('Stewie Griffin', '.', ''),
              repeat('#',26),
              'abcdefghijklmnopqrstuvwxyz'), '#', '')
from t1
```

```
REP
---
S G
```

На следующем шаге пробел заменяется точкой, опять же с помощью функции REPLACE:

```
select replace(
    replace(
        translate(replace('Stewie Griffin', '.', ''),
                    repeat('#',26),
                    'abcdefghijklmnopqrstuvwxyz'), '#', ''), ' ', '.') || '.'
from t1
```

```
REPLA
-----
S.G
```

В завершение в конец строки инициалов добавляется точка.

Oracle и PostgreSQL

Функция REPLACE удаляет из имени все символы точки (для обработки второго инициала), а функция TRANSLATE преобразовывает все строчные буквы в символ #:

```
select translate(replace('Stewie Griffin', '.', ''),
                'abcdefghijklmnopqrstuvwxyz',
                rpad('#',26,'#'))
from t1
```

```
TRANSLATE('STE
-----
S##### G#####
```

В полученном результате инициалами являются все символы, отличные от #. Далее функция REPLACE удаляет все символы #:

```
select replace(
    translate(replace('Stewie Griffin','.', ''),
        'abcdefghijklmnopqrstuvwxyz',
        rpad('#',26,'#')), '#', '')
from t1
```

```
REP
---
S G
```

На следующем шаге пробел заменяется точкой, опять же с помощью функции REPLACE:

```
select replace(
    replace(
        translate(replace('Stewie Griffin','.', ''),
            'abcdefghijklmnopqrstuvwxyz',
            rpad('#',26,'#') ), '#', ''), ' ', '.') || '.'
from t1
```

```
REPLA
-----
S.G
```

В завершение в конец строки инициалов добавляется точка.

MySQL

Символы точки из имени удаляются с помощью вложенного представления Y. Чтобы определить требуемое количество вызовов функции SUBSTR для извлечения инициалов, определяем количество пробелов в имени с помощью вложенного представления X. По местонахождению пробелов три вызова функции SUBSTRING_INDEX разбирают строку имени на составные части. Так как строка содержит только имя и фамилию, исполняется код блока ELSE выражения CASE:

```
select substr(substring_index(name, ' ',1),1,1) as a,
    substr(substring_index(name, ' ',-1),1,1) as b
from (select 'Stewie Griffin' as name from t1) x
```

```
A B
- -
S G
```

Если бы строка имени содержала отчество или инициал отчества, то соответствующий инициал был бы возвращен исполнением такого кода:

```
substr(name, length(substring_index(name, ' ',1))+2,1)
```

Этот код определяет окончание имени, а затем переходит на две позиции вперед к началу отчества или его инициала, т. е. к начальной позиции для исполнения

функции `SUBSTR`. Так как выбирается только один символ, из любой из этих подстрок возвращается инициал отчества. Полученные инициалы передаются функции `CONCAT_WS`, которая разделяет их точками:

```
select concat_ws('.',
                substr(substring_index(name, ' ',1),1,1),
                substr(substring_index(name, ' ',-1),1,1),
                '.' ) a
   from (select 'Stewie Griffin' as name from t1) x
```

```
A
-----
S.G..
```

В завершение удаляем из строки инициалов лишнюю точку.

6.8. Сортировка по подстрокам

ЗАДАЧА

Требуется упорядочить результирующее множество по определенной подстроке. Например, следующие записи:

```
ENAME
-----
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
CLARK
SCOTT
KING
TURNER
ADAMS
JAMES
FORD
MILLER
```

нужно упорядочить по *последним* двум символам каждой записи:

```
ENAME
-----
ALLEN
TURNER
MILLER
JONES
JAMES
MARTIN
BLAKE
```

```
ADAMS
KING
WARD
FORD
CLARK
SMITH
SCOTT
```

РЕШЕНИЕ

Ключ к решению этой задачи — определить, какую встроенную функцию используемой СУБД применить для извлечения подстроки, по которой будет выполняться сортировка. Обычно для этого используется функция `SUBSTR`.

DB2, Oracle, MySQL и PostgreSQL

Сортировка по заданной подстроке осуществляется посредством комбинации встроенных функций `LENGTH` и `SUBSTR`:

```
1 select ename
2   from emp
3  order by substr(ename, length(ename)-1,)
```

SQL Server

Сортировка по заданной подстроке осуществляется посредством комбинации встроенных функций `SUBSTRING` и `LEN`:

```
1 select ename
2   from emp
3  order by substring(ename, len(ename)-1, 2)
```

Обсуждение

Выбрать любую часть строки, по которой будет выполняться сортировка результирующего множества, можно с помощью выражения `SUBSTR` в конструкции `ORDER BY`. Но это не единственный способ решения этой задачи, и строки можно отсортировать по результатам практически любого другого выражения.

6.9. Сортировка по числу в строке

ЗАДАЧА

Требуется упорядочить результирующее множество по числу, входящему в состав строк. Например, следующее представление:

```
create view V as
select e.ename ||' '||
       cast(e.empno as char(4))||' '||
       d.dname as data
```

```
from emp e, dept d
where e.deptno=d.deptno
```

возвращает такие данные:

```
DATA
-----
CLARK 7782 ACCOUNTING
KING 7839 ACCOUNTING
MILLER 7934 ACCOUNTING
SMITH 7369 RESEARCH
JONES 7566 RESEARCH
SCOTT 7788 RESEARCH
ADAMS 7876 RESEARCH
FORD 7902 RESEARCH
ALLEN 7499 SALES
WARD 7521 SALES
MARTIN 7654 SALES
BLAKE 7698 SALES
TURNER 7844 SALES
JAMES 7900 SALES
```

Нам нужно упорядочить это результирующее множество по номеру служащего, расположенному в строке между именем служащего и названием его отдела:

```
DATA
-----
SMITH 7369 RESEARCH
ALLEN 7499 SALES
WARD 7521 SALES
JONES 7566 RESEARCH
MARTIN 7654 SALES
BLAKE 7698 SALES
CLARK 7782 ACCOUNTING
SCOTT 7788 RESEARCH
KING 7839 ACCOUNTING
TURNER 7844 SALES
ADAMS 7876 RESEARCH
JAMES 7900 SALES
FORD 7902 RESEARCH
MILLER 7934 ACCOUNTING
```

РЕШЕНИЕ

Во всех последующих решениях используются функции и синтаксис соответствующей СУБД, но при этом базовый подход одинаков для них всех — использование встроенных функций `REPLACE` и `TRANSLATE`. Суть подхода состоит в удалении с помощью этих функций нецифровых символов из строк, оставляя в них только цифровые значения, по которым и будет выполняться сортировка.

DB2

Для сортировки по цифровым символам используем встроенные функции REPLACE и TRANSLATE:

```

1 select data
2   from V
3  order by
4         cast(
5         replace(
6         translate(data,repeat('#',length(data)),
7         replace(
8         translate(data,'#####', '0123456789'),
9         '#',''), '#','') as integer)

```

Oracle

Для сортировки по цифровым символам используем встроенные функции REPLACE и TRANSLATE:

```

1 select data
2   from V
3  order by
4         to_number(
5         replace(
6         translate(data,
7         replace(
8         translate(data,'0123456789','#####'),
9         '#'),rpad('#',20,'#')),'#'))

```

PostgreSQL

Для сортировки по цифровым символам используем встроенные функции REPLACE и TRANSLATE:

```

1 select data
2   from V
3  order by
4         cast(
5         replace(
6         translate(data,
7         replace(
8         translate(data,'0123456789','#####'),
9         '#',''),rpad('#',20,'#')),'#','') as integer)

```

MySQL

На момент подготовки этой книги MySQL не поддерживает функцию TRANSLATE.

Обсуждение

Представление `V` создается лишь с целью предоставления записей для демонстрации решения этого рецепта. В нем просто конкатенируется несколько столбцов таблицы `EMP`. В решении демонстрируется сортировка таких составных строк по входящему в них номеру служащего.

Оператор `ORDER BY` в решениях может выглядеть несколько устрашающе, но если его исследовать по составным частям, он становится легко понятным. Сортировку строк по числовым подстрокам легче всего выполнить, предварительно удалив из них нечисловые символы. Затем оставшиеся числовые символы строк приводятся к числовому типу, после чего строки можно сортировать в любом требуемом порядке. Прежде чем рассматривать работу каждой функции, важно разобраться, почему они вызываются в таком порядке. Начиная с самого внутреннего вызова, функции `TRANSLATE` (строка 8 кода каждого решения), видим следующее:

1. Вызывается функция `TRANSLATE` (строка 8), и результаты возвращаются в...
2. Функцию `REPLACE` (строка 7), и ее результаты возвращаются в...
3. Функцию `TRANSLATE` (строка 6), и ее результаты возвращаются в...
4. Функцию `REPLACE` (строка 5), и ее результаты возвращаются в...
5. Функцию `CAST`, чтобы вернуть результат числового типа.

Первым делом нужно преобразовать числа в символы, отсутствующие в строке. Мы выбрали символ `#` и с помощью функции `TRANSLATE` преобразовали в него все числовые символы строки. Результаты такого преобразования показаны в выводе следующего запроса справа (слева показаны исходные данные):

```
select data,
       translate(data, '0123456789', '#####') as tmp
from V
```

DATA	TMP
CLARK 7782 ACCOUNTING	CLARK ##### ACCOUNTING
KING 7839 ACCOUNTING	KING ##### ACCOUNTING
MILLER 7934 ACCOUNTING	MILLER ##### ACCOUNTING
SMITH 7369 RESEARCH	SMITH ##### RESEARCH
JONES 7566 RESEARCH	JONES ##### RESEARCH
SCOTT 7788 RESEARCH	SCOTT ##### RESEARCH
ADAMS 7876 RESEARCH	ADAMS ##### RESEARCH
FORD 7902 RESEARCH	FORD ##### RESEARCH
ALLEN 7499 SALES	ALLEN ##### SALES
WARD 7521 SALES	WARD ##### SALES
MARTIN 7654 SALES	MARTIN ##### SALES
BLAKE 7698 SALES	BLAKE ##### SALES
TURNER 7844 SALES	TURNER ##### SALES
JAMES 7900 SALES	JAMES ##### SALES

Функция `TRANSLATE` обнаруживает все числовые символы в строке и преобразовывает их в символ `#`. Модифицированные таким образом строки передаются функции `REPLACE` (строка 7), которая удаляет из них все символы `#`:

```
select data,
       replace(
         translate(data, '0123456789', '#####'), '#') as tmp
from V
```

DATA	TMP
CLARK 7782 ACCOUNTING	CLARK ACCOUNTING
KING 7839 ACCOUNTING	KING ACCOUNTING
MILLER 7934 ACCOUNTING	MILLER ACCOUNTING
SMITH 7369 RESEARCH	SMITH RESEARCH
JONES 7566 RESEARCH J	ONES RESEARCH
SCOTT 7788 RESEARCH	SCOTT RESEARCH
ADAMS 7876 RESEARCH	ADAMS RESEARCH
FORD 7902 RESEARCH	FORD RESEARCH
ALLEN 7499 SALES	ALLEN SALES
WARD 7521 SALES	WARD SALES
MARTIN 7654 SALES	MARTIN SALES
BLAKE 7698 SALES	BLAKE SALES
TURNER 7844 SALES	TURNER SALES
JAMES 7900 SALES	JAMES SALES

Далее результаты функции `REPLACE` снова передаются функции `TRANSLATE`, но на этот раз во внешнем вызове этой функции в решении. Функция `TRANSLATE` выполняет в исходной строке поиск любых символов, совпадающих с символами в соответствующей строке таблицы `TMP`, и при обнаружении таких преобразовывает их в символы `#`.

Благодаря этому преобразованию все нечисловые символы можно рассматривать как один символ (поскольку они все заменяются одинаковым символом):

```
select data, translate(data,
                      replace(
                        translate(data, '0123456789', '#####'),
                        '#'),
                      rpad('#', length(data), '#')) as tmp
from V
```

DATA	TMP
CLARK 7782 ACCOUNTING	#####7782#####
KING 7839 ACCOUNTING	#####7839#####
MILLER 7934 ACCOUNTING	#####7934#####
SMITH 7369 RESEARCH	#####7369#####
JONES 7566 RESEARCH	#####7566#####


```

SCOTT 7788 RESEARCH      #####7788#####
ADAMS 7876 RESEARCH      #####7876#####
FORD 7902 RESEARCH       #####7902#####
ALLEN 7499 SALES         #####7499#####
WARD 7521 SALES          #####7521#####
MARTIN 7654 SALES        #####7654#####
BLAKE 7698 SALES         #####7698#####
TURNER 7844 SALES        #####7844#####
JAMES 7900 SALES         #####7900#####

```

Далее удаляем из строки все символы #, вызывая для этого функцию REPLACE (строка 5), оставляя только числовые символы:

```

select data, replace(
    translate(data,
        replace(
            translate(data, '0123456789', '#####'),
                '#'),
            rpad('#', length(data), '#')), '#') as tmp
from V

```

DATA	TMP
CLARK 7782 ACCOUNTING	7782
KING 7839 ACCOUNTING	7839
MILLER 7934 ACCOUNTING	7934
SMITH 7369 RESEARCH	7369
JONES 7566 RESEARCH	7566
SCOTT 7788 RESEARCH	7788
ADAMS 7876 RESEARCH	7876
FORD 7902 RESEARCH	7902
ALLEN 7499 SALES	7499
WARD 7521 SALES	7521
MARTIN 7654 SALES	7654
BLAKE 7698 SALES	7698
TURNER 7844 SALES	7844
JAMES 7900 SALES	7900

Наконец, приводим (строка 4) значения строк TMP к числовому типу, используя для этого соответствующую встроенную функцию СУБД (обычно CAST):

```

select data, to_number(
    replace(
        translate(data,
            replace(
                translate(data, '0123456789', '#####'),
                    '#'),
                rpad('#', length(data), '#')), '#')) as tmp
from V

```

DATA	TMP
CLARK 7782 ACCOUNTING	7782
KING 7839 ACCOUNTING	7839
MILLER 7934 ACCOUNTING	7934
SMITH 7369 RESEARCH	7369
JONES 7566 RESEARCH	7566
SCOTT 7788 RESEARCH	7788
ADAMS 7876 RESEARCH	7876
FORD 7902 RESEARCH	7902
ALLEN 7499 SALES	7499
WARD 7521 SALES	7521
MARTIN 7654 SALES	7654
BLAKE 7698 SALES	7698
TURNER 7844 SALES	7844
JAMES 7900 SALES	7900

При разработке подобных запросов полезно размещать выражения в списке `SELECT`. Это позволяет просматривать промежуточные результаты в процессе создания конечного решения. Но поскольку целью этого рецепта является упорядочивание результатов, то в конечном итоге все вызовы функций следует поместить в оператор `ORDER BY`:

```
select data
  from v
 order by
    to_number(
      replace(
        translate( data,
          replace(
            translate( data, '0123456789', '#####'),
              '#'), rpad('#', length(data), '#')), '#'))
```

```
DATA
-----
SMITH 7369 RESEARCH
ALLEN 7499 SALES
WARD 7521 SALES
JONES 7566 RESEARCH
MARTIN 7654 SALES
BLAKE 7698 SALES
CLARK 7782 ACCOUNTING
SCOTT 7788 RESEARCH
KING 7839 ACCOUNTING
TURNER 7844 SALES
ADAMS 7876 RESEARCH
JAMES 7900 SALES
FORD 7902 RESEARCH
MILLER 7934 ACCOUNTING
```

Напоследок отметим, что данные, как можно видеть, состоят из трех полей, только одно из которых является числовым. В случае нескольких числовых полей их нужно было бы конкатенировать в одно число, прежде чем выполнять сортировку строк.

6.10. Создание из строк таблицы списка с разделителями

ЗАДАЧА

Требуется вернуть значения строк таблицы не в виде строк с вертикальными столбцами, как обычно, а в виде списка значений столбцов, разделенных, например, запятыми. То есть преобразовать результирующее множество такого вида:

```
DEPTNO EMPS
-----
10 CLARK
10 KING
10 MILLER
20 SMITH
20 ADAMS
20 FORD
20 SCOTT
20 JONES
30 ALLEN
30 BLAKE
30 MARTIN
30 JAMES
30 TURNER
30 WARD
```

ВОТ В ТАКОЕ:

```
DEPTNO EMPS
-----
10 CLARK, KING, MILLER
20 SMITH, JONES, SCOTT, ADAMS, FORD
30 ALLEN, WARD, MARTIN, BLAKE, TURNER, JAMES
```

РЕШЕНИЕ

Решение такой задачи требует индивидуального подхода для каждой рассматриваемой здесь СУБД. Ключом же к решению является применение встроенных функций конкретной СУБД. Понимание доступных средств своей СУБД позволит вам использовать ее функциональность для креативного решения задачи, которая обычно не входит в круг задач, ставящихся перед SQL.

В настоящее время большинство СУБД оснащены специальной функцией для конкатенирования строк. Например, MySQL поддерживает функцию `GROUP_CONCAT` (начиная с самых ранних версий), а SQL Server — `STRING_ADD` (начиная с SQL Ser-

ver 2017). Такие функции имеют похожий синтаксис и позволяют легко и быстро решить эту задачу.

DB2

Для создания списка с разделителями используем встроенную функцию LIST_AGG:

```
1 select deptno,  
2     list_agg(ename ',') within GROUP(Order by 0) as emps  
3   from emp  
4  group by deptno
```

MySQL

Для создания списка с разделителями используем встроенную функцию GROUP_CONCAT:

```
1 select deptno,  
2     group_concat(ename order by empno separator, ',') as emps  
3   from emp  
4  group by deptno
```

Oracle

Для создания списка с разделителями используем встроенную функцию SYS_CONNECT_BY_PATH:

```
1 select deptno,  
2     ltrim(sys_connect_by_path(ename, ','), ',') emps  
3   from (  
4 select deptno,  
5     ename,  
6     row_number() over  
7       (partition by deptno order by empno) rn,  
8     count(*) over  
9       (partition by deptno) cnt  
10  from emp  
11  )  
12 where level = cnt  
13 start with rn = 1  
14 connect by prior deptno = deptno and prior rn = rn-1
```

PostgreSQL и SQL Server

Для создания списка с разделителями используем встроенную функцию STRING_AGG:

```
1 select deptno,  
2     string_agg(ename order by empno separator, ',') as emps  
3   from emp  
4  group by deptno
```

Обсуждение

Научиться создавать списки с разделителями в SQL весьма полезным, поскольку такая задача возникает достаточно часто. В стандарте SQL:2016 для решения этой задачи была введена функция `LIST_AGG`, но реализована пока что она только в DB2. К счастью, другие СУБД поддерживают подобные функции, часто с более простым синтаксисом.

MySQL

Функция `GROUP_CONCAT` конкатенирует переданные ей значения столбца, в нашем случае — столбца `ENAME`. Поскольку это агрегатная функция, запрос должен содержать оператор `GROUP BY`.

PostgreSQL и SQL Server

Синтаксис функции `STRING_AGG` похож на синтаксис функции `GROUP_CONCAT` до такой степени, что запрос для DB2 можно использовать и для этих СУБД, просто заменив `GROUP_CONCAT` на `STRING_AGG`.

Oracle

Первый шаг к пониманию работы запроса для Oracle — разбить его на составные части. Исполнив отдельно запрос вложенного представления (строки 4–10), получим результирующее множество, содержащее следующие данные для каждого служащего: номер отдела, имя служащего, ранг служащего в отделе, определяемый положением его номера `EMPNO` в упорядоченном по возрастанию списке номеров, а также общее количество служащих в этом отделе. Например:

```
select deptno,
       ename,
       row_number() over
         (partition by deptno order by empno) rn,
       count(*) over (partition by deptno) cnt
from emp
```

DEPTNO	ENAME	RN	CNT
10	CLARK	1	3
10	KING	2	3
10	MILLER	3	3
20	SMITH	1	5
20	JONES	2	5
20	SCOTT	3	5
20	ADAMS	4	5
20	FORD	5	5
30	ALLEN	1	6
30	WARD	2	6
30	MARTIN	3	6

Ранг служащего (обозначен в запросе псевдонимом `RN`) нужен для того, чтобы позволить обход дерева. Поскольку функция `ROW_NUMBER` создает список, начиная с единицы, без дубликатов и пропусков значений, для обращения к предыдущей (родительской) строке нужно просто вычесть 1 из текущего значения. Например, номер перед 3 равен 3 минус 1, т. е., 2. В рассматриваемом контексте, 2 — это родитель 3, как можно видеть в строке 12 кода. Кроме этого, следующие линии кода:

```
start with rn = 1
connect by prior deptno = deptno
```

идентифицируют корень каждого `DEPTNO`, как `RN` со значением 1, и создают новый список при обнаружении нового отдела (т. е. при обнаружении `RN` со значением 1).

На этом этапе важно рассмотреть более подробно блок `ORDER BY` функции `ROW_NUMBER`. Не забывайте, что ранг служащих определяется их номером `EMPNO`, и список имен создается в порядке этих номеров. Вычисляется значение количества служащих в отделе (псевдоним `CNT`), которое используется для обеспечения возврата запросом только списка, содержащего всех служащих отдела. Такой подход требуется потому, что `SYS_CONNECT_BY_PATH` создает список итеративно, и мы не хотим получить неполный список.

Для иерархических запросов начальное значение псевдостолбца `LEVEL` равно 1 (для запросов без использования `CONNECT BY` это значение равно 0, причем для версии 10 СУБД и более поздних версий столбец `LEVEL` доступен только при использовании `CONNECT BY`) и инкрементируется на единицу после обработки каждого служащего отдела (для каждого уровня вложенности иерархии). Благодаря этому мы знаем, что, когда значение `LEVEL` становится равным `CNT`, мы достигли последнего `EMPNO` и получим полный список.



Функция `SYS_CONNECT_BY_PATH` вставляет заданный разделитель (в нашем случае запятую) в начало списка, что желательно не всегда. В приведенном решении эта запятая удаляется функцией `LTRIM`.

6.11. Преобразование данных с разделителями в многозначный список оператора `IN`

ЗАДАЧА

Требуется передать данные с разделителями в итератор значений оператора `IN` конструкции `WHERE`. Возьмем, например, следующую строку:

```
7654,7698,7782,7788
```

Мы хотим использовать эту строку в конструкции `WHERE`, например, в следующем запросе:

```
select ename, sal, deptno
   from emp
  where empno in ( '7654,7698,7782,7788' )
```

Но попытка исполнить этот запрос возвратит ошибку, поскольку столбец `EMPNO` имеет числовой тип, а список значений `IN` состоит из одного значения строкового типа. Нам нужно добиться, чтобы эта строка обрабатывалась как список числовых значений, разделенных запятыми.

РЕШЕНИЕ

С первого взгляда может казаться, что SQL должен рассматривать строку с разделителями как список значений с разделителями, но в действительности этого не происходит. никоим образом SQL не может знать, что содержащиеся внутри кавычек запятые обозначают многозначный список, поскольку все содержимое кавычек интерпретируется им как одно строковое значение. Поэтому нам надо разбить строку на отдельные составляющие ее значения `EMPNO`. Для этого следует выполнить обход строки, но не посимвольный, а по значениям списка, разбив ее на действительные значения `EMPNO`.

DB2

Значения передаваемой в список оператора `IN` строки очень просто преобразовать в строки таблицы, выполнив обход строки. Справиться с этой задачей нам помогут функции `ROW_NUMBER`, `LOCATE` и `SUBSTR`:

```

1 select empno,ename,sal,deptno
2   from emp
3  where empno in (
4 select cast(substr(c,2,locate(',',c,2)-2) as integer) empno
5   from (
6 select substr(csv.emps,cast(iter.pos as integer)) as c
7   from (select ','||'7654,7698,7782,7788'||',' emp
8         from t1) csv,
9        (select id as pos
10       from t100 ) iter
11  where iter.pos <= length(csv.emps)
12        ) x
13  where length(c) > 1
14     and substr(c,1,1) = ','
15        )

```

MySQL

Значения передаваемой в список оператора `IN` строки очень просто преобразовать в строки таблицы, выполнив обход строки:

```

1 select empno, ename, sal, deptno
2   from emp
3  where empno in
4     (
5 select substring_index(
6     substring_index(list.vals,',',iter.pos),',',-1) empno

```

```

7  from (select id pos from t10) as iter,
8      (select '7654,7698,7782,7788' as vals
9      from t1) list
10 where iter.pos <=
11      (length(list.vals)-length(replace(list.vals,',','')))+1
12      )

```

Oracle

Значения передаваемой в список IN строки очень просто преобразовать в строки, выполнив обход строки. Справиться с этой задачей нам помогут функции ROWNUM, SUBSTR и INSTR:

```

1 select empno,ename,sal,deptno
2   from emp
3  where empno in (
4      select to_number(
5          rtrim(
6          substr(emps,
7              instr(emps,',',1,iter.pos)+1,
8              instr(emps,',',1,iter.pos+1) _
9              instr(emps,',',1,iter.pos)),',')) emp
10     from (select '||'7654,7698,7782,7788'||',' emp from t1) csv,
11          (select rownum pos from emp) iter
12     where iter.pos <= ((length(csv.emps)_
13         length(replace(csv.emps,',')))/length(','))_1
14 )

```

PostgreSQL

Значения передаваемой в список IN строки очень просто преобразовать в строки, выполнив обход строки. Разобрать строку на отдельные числовые значения можно с помощью функции SPLIT_PART:

```

1 select ename,sal,deptno
2   from emp
3  where empno in (
4  select cast(empno as integer) as empno
5   from (
6  select split_part(list.vals,',',iter.pos) as empno
7   from (select id as pos from t10) iter,
8      (select '||'7654,7698,7782,7788'||',' as vals
9      from t1) list
10  where iter.pos <=
11      length(list.vals)-length(replace(list.vals,',',''))
12      ) z
13  where length(empno) > 0
14      )

```


SQL Server

Значения передаваемой в список IN строки очень просто преобразовать в строки, выполнив обход строки. Справиться с этой задачей нам помогут функции ROW_NUMBER, CHARINDEX и SUBSTRING:

```

1 select empno,ename,sal,deptno
2   from emp
3  where empno in (select substring(c,2,charindex(',',c,2)-2) as empno
4   from (
5  select substring(csv.emps,iter.pos,len(csv.emps)) as c
6   from (select ','+'7654,7698,7782,7788'+',' as emps
7    from t1) csv,
8   (select id as pos
9    from t100) iter
10  where iter.pos <= len(csv.emps)
11   ) x
12  where len(c) > 1
13   and substring(c,1,1) = ','
14   )

```

Обсуждение

Первый и самый важный шаг этого решения — выполнить обход строки. Затем надо только выполнить парсинг строки на отдельные числовые значения посредством встроенных функций используемой СУБД.

DB2 и SQL Server

Обход строки осуществляется во вложенном представлении X (строки кода 6–11). Суть решения решения заключается в «проходе строки» таким образом, чтобы в каждой следующей строке количество символов было на один меньше, чем в предыдущей:

```

,7654,7698,7782,7788,
7654,7698,7782,7788,
654,7698,7782,7788,
54,7698,7782,7788,
4,7698,7782,7788,
,7698,7782,7788,
7698,7782,7788,
698,7782,7788,
98,7782,7788,
8,7782,7788,
,7782,7788,
7782,7788,
782,7788,
82,7788,

```

```
2,7788,
,7788,
7788,
788,
88,
8,
,
```

Обратите внимание, что заключение строки в запятые (разделитель) устраняет необходимость выполнять специальные проверки на начало или окончание строки.

Следующий шаг — оставить только те значения, которые требуется использовать в списке оператора `IN`. Эти значения начинаются с символа запятой, за исключением последней строки, состоящей из одной запятой. Строки с ведущей запятой определяем посредством функций `SUBSTR` или `SUBSTRING` и сохраняем все символы до следующей запятой в текущей строке. После этого выполняем приведение полученного значения к числовому типу, чтобы его можно было сравнить со значением числового столбца `EMPNO` (строки кода 4–14):

```
EMPNO
-----
7654
7698
7782
7788
```

В завершение используем полученные результаты в подзапросе для получения требуемых строк.

MySQL

Обход строки осуществляется во вложенном представлении (строки кода 5–9). Выражение в строках 10–11 определяет количество значений в исходной строке, подсчитывая обнаруженные запятые (разделители) и добавляя к полученному числу 1. Функция `SUBSTRING_INDEX` (строка 6) возвращает все символы строки слева от n -й запятой (разделителя):

```
+-----+
| empno |
+-----+
| 7654  |
| 7654,7698 |
| 7654,7698,7782 |
| 7654,7698,7782,7788 |
+-----+
```

Эти строки затем передаются в новый вызов функции `SUBSTRING_INDEX` (строка 5). На этот раз n -й разделитель меньше на 1, в результате чего сохраняются все значения справа от n -го разделителя:

```
+-----+
| empno |
+-----+
| 7654  |
| 7698  |
| 7782  |
| 7788  |
+-----+
```

В завершение полученные результаты вставляются в подзапрос.

Oracle

Первым шагом выполняем обход строки:

```
select emps,pos
      from (select ','||'7654,7698,7782,7788'||',' emps
            from t1) csv,
          (select rownum pos from emp) iter
 where iter.pos <=
        ((length(csv.emps)-length(replace(csv.emps,', ')))/length(', '))-1
```

EMPS	POS
,7654,7698,7782,7788,	1
,7654,7698,7782,7788,	2
,7654,7698,7782,7788,	3
,7654,7698,7782,7788,	4

Количество возвращенных строк представляет количество значений в списке. Значения POS критически важны для запроса, т. к. они требуются для выполнения парсинга строки на отдельные значения. Парсинг строк осуществляется посредством функций SUBSTR и INSTR. Для определения *n*-го делителя в каждой строке используется значение POS. Заключение строк в запятое (разделитель) устраняет необходимость выполнять специальные проверки на начало или окончание строки. Значения, передаваемые функциям SUBSTR и INSTR (строки 7–9), определяют *n*-й и *n*+1-й разделители. Вычитая значение, возвращенное для текущей запятой (номер позиции текущей запятой в строке), из значения, возвращенного для следующей запятой (номер позиции следующей запятой в строке), можно извлечь из строки все составляющие ее значения:

```
select substr(emps,
             instr(emps,',',1,iter.pos)+1,
             instr(emps,',',1,iter.pos+1)
             -instr(emps,',',1,iter.pos)) emps
      from (select ','||'7654,7698,7782,7788'||',' emps
            from t1) csv,
          (select rownum pos from emp) iter
 where iter.pos <=
        ((length(csv.emps)-length(replace(csv.emps,', ')))/length(', '))-1
```

```
EMPS
-----
7654,
7698,
7782,
7788,
```

В завершение удаляем из каждого значения конечную запятую, приводим его к числовому типу и вставляем его в подзапрос.

PostgreSQL

Обход строки осуществляется во вложенном представлении Z (строки кода 6–9). Количество возвращаемых строк определяется количеством значений в исходной строке. А количество значений в строке определяется вычитанием количества символов в строке без разделителей из этого же количества, но только с разделителями (строка 9). Парсинг строки для разбиения ее на составные значения осуществляется функцией `SPLIT_PART`, которая ищет значение перед *n*-м разделителем:

```
select list.vals,
       split_part(list.vals,',',iter.pos) as empno,
       iter.pos
  from (select id as pos from t10) iter,
       (select ',|||'7654,7698,7782,7788'||',' as vals
        from t1) list
 where iter.pos <=
        length(list.vals)-length(replace(list.vals,',',''))
```

vals	empno	pos
,7654,7698,7782,7788,		1
,7654,7698,7782,7788,	7654	2
,7654,7698,7782,7788,	7698	3
,7654,7698,7782,7788,	7782	4
,7654,7698,7782,7788,	7788	5

В завершение приводим значения (`EMPNO`) к числовому типу и вставляем их в подзапрос.

6.12. Упорядочение строки по алфавиту

ЗАДАЧА

Требуется отсортировать символы строк таблицы в алфавитном порядке. Например, возьмем следующее результирующее множество с одним столбцом:

```
ENAME
-----
ADAMS
ALLEN
BLAKE
```

CLARK
 FORD
 JAMES
 JONES
 KING
 MARTIN
 MILLER
 SCOTT
 SMITH
 TURNER
 WARD

Мы хотим получить из него следующий результат с двумя столбцами:

OLD_NAME	NEW_NAME
ADAMS	AADMS
ALLEN	AELLN
BLAKE	ABEKL
CLARK	ACKLR
FORD	DFOR
JAMES	AEJMS
JONES	EJNOS
KING	GIKN
MARTIN	AIMNRT
MILLER	EILLMR
SCOTT	COSTT
SMITH	HIMST
TURNER	ENRRTU
WARD	ADRW

РЕШЕНИЕ

Эта задача хорошо иллюстрирует, как улучшенная стандартизация способствует разработке более похожих и, следовательно, переносимых решений.

DB2

Чтобы отсортировать строки по алфавиту, необходимо выполнить обход каждой строки, а затем упорядочить ее символы:

```

1 select ename,
2     listagg(c, '') WITHIN GROUP( ORDER BY c)
3   from (
4     select a.ename,
5     substr(a.ename, iter.pos, 1
6     ) as c
7   from emp a,
8     (select id as pos from t10) iter
9     where iter.pos <= length(a.ename)
```

```

10      order by 1,2
11      ) x
12      Group By c

```

MySQL

Ключ к этому решению — функция `GROUP_CONCAT`, которая позволяет не только конкатенировать составляющие символы каждого имени, но также упорядочить их:

```

1 select ename, group_concat(c order by c separator '')
2   from (
3 select ename, substr(a.ename,iter.pos,1) c
4   from emp a,
5        ( select id pos from t10 ) iter
6  where iter.pos <= length(a.ename)
7        ) x
8  group by ename

```

Oracle

Функция `SYS_CONNECT_BY_PATH` позволяет создать список посредством итераций:

```

1 select old_name, new_name
2   from (
3 select old_name, replace(sys_connect_by_path(c,' '), ' ') new_name
4   from (
5 select e.ename old_name,
6        row_number() over(partition by e.ename
7                          order by substr(e.ename,iter.pos,1)) rn,
8        substr(e.ename,iter.pos,1) c
9   from emp e,
10        ( select rownum pos from emp ) iter
11  where iter.pos <= length(e.ename)
12  order by 1
13        ) x
14  start with rn = 1
15 connect by prior rn = rn-1 and prior old_name = old_name
16        )
17  where length(old_name) = length(new_name)

```

PostgreSQL

Начиная с версии 9.0, для этой СУБД доступна функция `STRING_AGG`, позволяющая упорядочивать символы строки:

```

      select ename, string_agg(c , ''
                                ORDER BY c)
from (
      select a.ename,
            substr(a.ename,iter.pos,1) as c

```

```

    from emp a,
         (select id as pos from t10) iter
   where iter.pos <= length(a.ename)
   order by 1,2
         ) x
   Group By c

```

SQL Server

Для SQL Server 2017 и более поздних версий применимо использующее функцию `STRING_AGG` решение, приведенное для PostgreSQL. Чтобы отсортировать строки по алфавиту в более ранних версиях, необходимо выполнить обход каждой строки, а затем упорядочить ее символы:

```

1 select ename,
2         max(case when pos=1 then c else '' end)+
3         max(case when pos=2 then c else '' end)+
4         max(case when pos=3 then c else '' end)+
5         max(case when pos=4 then c else '' end)+
6         max(case when pos=5 then c else '' end)+
7         max(case when pos=6 then c else '' end)
8   from (
9   select e.ename,
10         substring(e.ename,iter.pos,1) as c,
11         row_number() over (
12         partition by e.ename
13         order by substring(e.ename,iter.pos,1)) as pos
14   from emp e,
15        (select row_number()over(order by ename) as pos
16         from emp) iter
17  where iter.pos <= len(e.ename)
18         ) x
19  group by ename

```

Обсуждение

SQL Server

Вложенный запрос X возвращает каждый символ имени в виде строки. Функция `SUBSTR` (или `SUBSTRING`) извлекает все символы из имен, а функция `ROW_NUMBER` ранжирует каждый символ в алфавитном порядке:

```

ENAME C POS
----- - ---
ADAMS A  1
ADAMS A  2
ADAMS D  3
ADAMS M  4
ADAMS S  5
...

```

Чтобы вернуть каждую букву строки в виде строки таблицы, необходимо выполнить обход исходной строки. Это осуществляется посредством вложенного запроса `ITER`.

Отсортировав буквы всех имен по алфавиту, в завершение складываем их обратно в строку, но уже в отсортированном порядке. Позиция каждой буквы определяется выражениями `CASE` (строки 2–7). При нахождении символа в определенной позиции осуществляется его конкатенация с результатом следующего вычисления (следующего выражения `CASE`). Поскольку также используется агрегатная функция `MAX`, для каждой позиции `POS` возвращается только один символ, в результате чего для каждого имени возвращается только одна строка. Вычисление `CASE` продолжается до значения 6, т. е. до максимального количества символов любого имени в таблице `EMP`.

MySQL

Вложенное представление `X` (строки 3–6) возвращает каждый символ имени в отдельной строке. Символы из имени извлекаются функцией `SUBSTR`:

```
ENAME C
-----
ADAMS A
ADAMS A
ADAMS D
ADAMS M
ADAMS S
...
```

Обход строки осуществляется вложенным запросом `ITER`. Вся остальная работа выполняется функцией `GROUP_CONCAT`, которая не просто конкатенирует все буквы, но делает это в указанном алфавитном порядке.

Oracle

Здесь основная работа выполняется вложенным запросом `X` (строки 5–11), который извлекает символы каждого имени и располагает их в алфавитном порядке. В частности, осуществляется посимвольный обход строки с последующим упорядочиванием извлеченных символов. В остальной части запроса просто выполняется сборка символов в новом порядке.

Разобранные по символам имена можно просмотреть, исполнив отдельно вложенный запрос `X`:

```
OLD_NAME          RN C
-----
ADAMS             1 A
ADAMS             2 A
ADAMS             3 D
ADAMS             4 M
ADAMS             5 S
```


Следующий шаг — собрать упорядоченные по алфавиту отдельные символы имени в одну строку в этом же порядке. Это осуществляется посредством функции `SYS_CONNECT_BY_PATH`, которая добавляет каждый следующий символ в конец строки ранее конкатенированных таким образом символов:

```

OLD_NAME      NEW_NAME
-----
ADAMS         A
ADAMS         AA
ADAMS         AAD
ADAMS         AADM
ADAMS         AADMS

```

Последний шаг — отобразить упорядоченные по алфавиту строки только такой же длины, как и соответствующие исходные строки.

PostgreSQL

В этом решении обход строки осуществляется вложенным представлением `V`. В определении представления функция `SUBSTR` извлекает отдельные символы из имени, и представление возвращает следующий результат:

```

ENAME C
-----
ADAMS A
ADAMS A
ADAMS D
ADAMS M
ADAMS S
...

```

В представлении также осуществляется упорядочивание результатов по `ENAME` и соответствующих извлеченных букв по алфавиту. Вложенное представление `X` возвращает из представления `V` имена и соответствующие извлеченные упорядоченные символы, количество вхождений каждого символа в имя и его позицию (в алфавитном порядке):

```

ename | c | cnt | pos
-----+-----+-----
ADAMS | A | 2 | 1
ADAMS | A | 2 | 1
ADAMS | D | 1 | 3
ADAMS | M | 1 | 4
ADAMS | S | 1 | 5

```

Критически важными для решения являются дополнительные столбцы `CNT` и `POS`, возвращаемые вложенным представлением `X`. Столбец `POS` служит для ранжирования символов, а `CNT` — для определения количества вхождений каждого символа в имя. Последний шаг — определить позицию по алфавиту каждого извлеченного символа имени и собрать их всех в одну строку в этом порядке.

6.13. Идентификация числовых подстрок в строке

ЗАДАЧА

Имеется столбец, для которого определен символьный (`char`) тип данных. К сожалению, строки содержат как числовые, так буквенные символы, причем вместе в одной строке. Возьмем, например, следующее представление V:

```
create view V as
select replace(mixed,' ','') as mixed
  from (
select substr(ename,1,2)||
       cast(deptno as char(4))||
       substr(ename,3,2) as mixed
  from emp
 where deptno = 10
 union all
select cast(empno as char(4)) as mixed
  from emp
 where deptno = 20
 union all
select ename as mixed
  from emp
 where deptno = 30
 ) x
select * from v
```

```
MIXED
-----
CL10AR
KI10NG
MI10LL
7369
7566
7788
7876
7902
ALLEN
WARD
MARTIN
BLAKE
TURNER
JAMES
```

После обработки этих данных мы хотим вернуть только те исходные строки, которые состоят из числовых символов, — все значение строки или только часть ее. В последнем случае нужно удалить все нечисловые символы и вернуть толь-

ко числовые. Таким образом, для приведенного здесь образца данных мы хотим получить следующее результирующее множество:

```
MIXED
-----
      10
      10
      10
     7369
     7566
     7788
     7876
     7902
```

РЕШЕНИЕ

Для решения этой задачи, требующего манипулирования строками и отдельными символами, лучше всего подойдут функции `REPLACE` и `TRANSLATE`. Решение состоит в преобразовании всех числовых символов в какой-либо один символ, в результате чего любое число можно будет выделить и извлечь, ссылаясь на этот символ.

DB2

Для разделения символьных и числовых данных используем встроенные функции `TRANSLATE`, `REPLACE` и `POSSTR`. В представлении `V` необходимо вызывать функцию `CAST`, т. к. в противном случае создать представление не получится из-за ошибок при преобразовании типов. Функция `REPLACE` требуется для удаления лишних пробелов, образующихся вследствие приведения к типу `CHAR` фиксированной длины:

```
1 select mixed old,
2     cast(
3     case
4     when
5     replace(
6     translate(mixed,'9999999999','0123456789'),'9','') = ''
7     then
8     mixed
9     else replace(
10    translate(mixed,
11    repeat('#',length(mixed)),
12    replace(
13    translate(mixed,'9999999999','0123456789'),'9',''),
14    '#','')
15    end as integer ) mixed
16 from V
17 where posstr(translate(mixed,'9999999999','0123456789'),'9') > 0
```

MySQL

Для MySQL синтаксис запроса для создания представления V немного другой:

```
create view V as
select concat(
    substr(ename,1,2),
    replace(cast(deptno as char(4)), ' ', ''),
    substr(ename,3,2)
) as mixed
from emp
where deptno = 10
union all
select replace(cast(empno as char(4)), ' ', '')
from emp where deptno = 20
union all
select ename from emp where deptno = 30
```

Поскольку MySQL не поддерживает функцию TRANSLATE, необходимо выполнить посимвольный обход каждой строки:

```
1 select cast(group_concat(c order by pos separator '') as unsigned)
2     as MIXED1
3   from (
4 select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
5   from V,
6        ( select id pos from t10 ) iter
7  where iter.pos <= length(v.mixed)
8        and ascii(substr(v.mixed,iter.pos,1)) between 48 and 57
9        ) y
10  group by mixed
11  order by 1
```

Oracle

Для отделения числовых данных от буквенных используем встроенные функции TRANSLATE, REPLACE и INSTR. Вызывать функцию CAST в представлении V необязательно. Для удаления лишних пробелов, образующихся вследствие приведения к типу CHAR фиксированной длины, используем функцию REPLACE. При желании оставить явное преобразование типов в определении представления рекомендуется выполнять приведение к типу VARCHAR2:

```
1 select to_number (
2     case
3     when
4         replace(translate(mixed, '0123456789', '9999999999'), '9')
5         is not null
6     then
7         replace(
8         translate(mixed,
```

```

9         replace(
10        translate(mixed, '0123456789', '9999999999'), '9'),
11        rpad('#', length(mixed), '#')), '#')
12     else
13         mixed
14     end
15 ) mixed
16 from V
17 where instr(translate(mixed, '0123456789', '9999999999'), '9') > 0

```

PostgreSQL

Для отделения числовых данных от буквенных используем встроенные функции `TRANSLATE`, `REPLACE` и `STRPOS`. Вызывать функцию `CAST` в представлении `V` необязательно. Для удаления лишних пробелов, образующихся вследствие приведения к типу `CHAR` фиксированной длины, используем функцию `REPLACE`. При желании оставить явное преобразование типов в определении представления рекомендуется выполнять приведение к типу `VARCHAR2`:

```

1 select cast(
2     case
3     when
4         replace(translate(mixed, '0123456789', '9999999999'), '9', '')
5         is not null
6     then
7         replace(
8         translate(mixed,
9         replace(
10        translate(mixed, '0123456789', '9999999999'), '9', ''),
11        rpad('#', length(mixed), '#')), '#', '')
12     else
13         mixed
14     end as integer ) as mixed
15 from V
16 where strpos(translate(mixed, '0123456789', '9999999999'), '9') > 0

```

SQL Server

Определить строки, содержащие числовые символы, можно, используя встроенную функцию `ISNUMERIC` совместно с поиском с подстановочным символом. Но процесс извлечения числовых символов из строки не особенно эффективен по причине отсутствия встроенной функции `TRANSLATE`.

Обсуждение

Для решения этой задачи хорошо подходит функция `TRANSLATE`, поскольку она позволяет с легкостью идентифицировать и отделить числовые символы от буквенных. Особенность решения состоит в преобразовании всех числовых символов

в какой-либо один символ, в результате чего вместо поиска разных символов можно искать только один.

DB2, Oracle и PostgreSQL

Синтаксис решений для этих СУБД несколько различен, но базовый принцип остается таким же. В этом обсуждении мы будем рассматривать решение для PostgreSQL.

Основная работа выполняется функциями `TRANSLATE` и `REPLACE`. Чтобы получить конечное результирующее множество, нужно вызвать несколько функций, как показано в следующем запросе:

```
select mixed as orig,
translate(mixed, '0123456789', '9999999999') as mixed1,
replace(translate(mixed, '0123456789', '9999999999'), '9', '') as mixed2,
  translate(mixed,
    replace(
      translate(mixed, '0123456789', '9999999999'), '9', ''),
      rpad('#', length(mixed), '#')) as mixed3,
    replace(
      translate(mixed,
        replace(
          translate(mixed, '0123456789', '9999999999'), '9', ''),
          rpad('#', length(mixed), '#')), '#', '') as mixed4
from V
where strpos(translate(mixed, '0123456789', '9999999999'), '9') > 0
```

ORIG	MIXED1	MIXED2	MIXED3	MIXED4	MIXED5
CL10AR	CL99AR	CLAR	##10##	10	10
KI10NG	KI99NG	KING	##10##	10	10
MI10LL	MI99LL	MILL	##10##	10	10
7369	9999		7369	7369	7369
7566	9999		7566	7566	7566
7788	9999		7788	7788	7788
7876	9999		7876	7876	7876
7902	9999		7902	7902	7902

Прежде всего обратите внимание на то, что удаляются все строки, не содержащие хотя бы одного числового символа. Понять, как это осуществляется, можно, изучив каждый столбец в приведенном результирующем множестве. Строки, которые остаются, показаны в столбце `ORIG` — из значений этих строк и будет создано конечное результирующее множество. Первый шаг в процессе извлечения числовых символов — преобразование любого числового символа в цифру 9 (цифра 9 была выбрана произвольно, можно использовать любую другую цифру), применяя для этого функцию `TRANSLATE`. Результат этого преобразования отображен в столбце `MIXED1`. Теперь, когда все числовые символы были преобразованы в цифру 9, с ними можно обращаться, как с одним элементом. Далее удаляем из строк все числовые

символы, используя для этого функцию `REPLACE`. Так как на этом этапе все числовые символы заменены символом `9`, `REPLACE` просто ищет эти символы и удаляет их. Конечный результат работы этой функции представлен в столбце `MIXED2`. Значения этого столбца используются для получения значений в столбце `MIXED3`. Для этого они сравниваются со значениями в столбце `ORIG`. Если этот столбец содержит какой-либо символ из столбца `MIXED2`, посредством функции `TRANSLATE` этот символ преобразовывается в символ `#`. В результате столбец `MIXED3` содержит исходные значения строк, в которых все буквенные символы были преобразованы в один символ. Теперь, когда все буквенные символы преобразованы в символ `#`, с ними можно обращаться как с одним элементом. На следующем шаге используем функцию `REPLACE`, чтобы удалить все символы `#` из значений в столбце `MIXED3`. В результате получаем только строки из числовых символов, как показано в столбце `MIXED4`. Последний шаг — преобразовываем все строки в числовой тип. Такое поэтапное рассмотрение процесса исполнения запроса должно сделать понятным работу предиката `WHERE`. Значения из столбца `MIXED1` передаются функции `STRPOS`, которая при обнаружении символа `9` (позиции в строке, в которой находится первый символ `9`) должна вернуть значение больше `0`. Возвращение для строки значения больше `0` означает, что строка содержит как минимум один числовой символ, и ее нужно оставить.

MySQL

Первый шаг — выполнить посимвольный обход строки, чтобы определить числовые символы:

```
select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
      from v,
      ( select id pos from t10 ) iter
     where iter.pos <= length(v.mixed)
     order by 1,2
```

```
+-----+-----+-----+
| mixed | pos | c   |
+-----+-----+-----+
| 7369  | 1  | 7   |
| 7369  | 2  | 3   |
| 7369  | 3  | 6   |
| 7369  | 4  | 9   |
...
| ALLEN | 1  | A   |
| ALLEN | 2  | L   |
| ALLEN | 3  | L   |
| ALLEN | 4  | E   |
| ALLEN | 5  | N   |
...
| CL10AR | 1  | C   |
| CL10AR | 2  | L   |
```

```

| CL10AR | 3 | 1 |
| CL10AR | 4 | 0 |
| CL10AR | 5 | A |
| CL10AR | 6 | R |
+-----+-----+-----+

```

Теперь, когда с каждым символом строк можно работать по отдельности, нужно оставить только те строки, которые содержат хотя бы один числовой символ в столбце C:

```

select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
  from V,
       ( select id pos from t10 ) iter
 where iter.pos <= length(v.mixed)
       and ascii(substr(v.mixed,iter.pos,1)) between 48 and 57
 order by 1,2

```

```

+-----+-----+-----+
| mixed | pos | c |
+-----+-----+-----+
| 7369  | 1  | 7 |
| 7369  | 2  | 3 |
| 7369  | 3  | 6 |
| 7369  | 4  | 9 |
...
| CL10AR | 3  | 1 |
| CL10AR | 4  | 0 |
...
+-----+-----+-----+

```

На данном этапе все строки столбца C содержат только числовые символы. Далее конкатенируем все числовые символы, чтобы получить соответствующие строки, используя для этого функцию `GROUP_CONCAT`. Наконец, вызываем функцию `CAST`, чтобы преобразовать полученные строки в числовой тип. Конечный результат показан в столбце `MIXED1`:

```

select cast(group_concat(c order by pos separator '') as unsigned)
  as MIXED1
  from (
select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
  from V,
       ( select id pos from t10 ) iter
 where iter.pos <= length(v.mixed)
       and ascii(substr(x.mixed,iter.pos,1)) between 48 and 57
       ) y
 group by mixed
 order by 1

```



```
+-----+
| MIXED1 |
+-----+
|      10 |
|      10 |
|      10 |
|     7369 |
|     7566 |
|     7788 |
|     7876 |
|     7902 |
+-----+
```

Напоследок следует заметить, что все содержащиеся в строке числовые символы будут конкатенироваться в одно числовое значение. Например, если строка содержит, например, значение 99Gennick87, то будет возвращено значение 9987. Это обстоятельство нужно иметь в виду, особенно при работе с сериализованными данными.

6.14. Извлечение n -й подстроки из списка с разделителями

ЗАДАЧА

Требуется извлечь указанную подстроку из списка с разделителями. Например, следующий запрос создает представление V с исходными данными для задачи:

```
create view V as
select 'mo,larry,curly' as name
       from t1
       union all
select 'tina,gina,jaunita,regina,leena' as name
       from t1
```

Содержимое этого представления будет следующим:

```
select * from v

NAME
-----
mo,larry,curly
tina,gina,jaunita,regina,leena
```

Нам нужно извлечь из каждой строки второе значение, чтобы получить следующее результирующее множество:

```
SUB
----
larry
gina
```

РЕШЕНИЕ

Ключ к решению этой задачи — вернуть каждое имя списка в отдельной строке, одновременно сохраняя порядок имен в списке. Конкретная реализация решения зависит от используемой СУБД.

DB2

Разбив посредством обхода строки каждую исходную строку `NAME` представления `V` на составляющие ее подстроки, применяем функцию `ROW_NUMBER`, чтобы вернуть только вторую подстроку каждой исходной строки:

```

1 select substr(c,2,locate(',',c,2)-2)
2   from (
3 select pos, name, substr(name, pos) c,
4        row_number() over( partition by name
5                          order by length(substr(name,pos)) desc) rn
6   from (
7 select ',' || csv.name || ',' as name,
8        cast(iter.pos as integer) as pos
9   from V csv,
10        (select row_number() over() pos from t100 ) iter
11  where iter.pos <= length(csv.name)+2
12        ) x
13  where length(substr(name,pos)) > 1
14        and substr(substr(name,pos),1,1) = ','
15        ) y
16  where rn = 2

```

MySQL

Разбив посредством обхода строки каждую исходную строку `NAME` представления `V` на составляющие ее подстроки, возвращаем только вторую подстроку каждой исходной строки, определяя ее по позициям запятых:

```

1 select name
2   from (
3 select iter.pos,
4        substring_index(
5        substring_index(src.name,',',iter.pos),',',-1) name
6   from V src,
7        (select id pos from t10) iter,
8  where iter.pos <=
9        length(src.name)-length(replace(src.name,',',''))
10        ) x
11  where pos = 2

```

Oracle

Разбив посредством обхода строки каждую исходную строку `NAME` представления `V` на составляющие ее подстроки, применяем функции `SUBSTR` и `INSTR`, чтобы вернуть только вторую подстроку каждой исходной строки:

```

1 select sub
2   from (
3 select iter.pos,
4        src.name,
5        substr( src.name,
6               instr( src.name, ',', 1, iter.pos )+1,
7               instr( src.name, ',', 1, iter.pos+1 ) -
8               instr( src.name, ',', 1, iter.pos )-1) sub
9   from (select ' ,||name||' as name from V) src,
10        (select rownum pos from emp) iter
11  where iter.pos < length(src.name)-length(replace(src.name, ','))
12        )
13  where pos = 2

```

PostgreSQL

Возвращаем каждое имя исходной строки в отдельной строке таблицы, используя для этого функцию `SPLIT_PART`:

```

1 select name
2   from (
3 select iter.pos, split_part(src.name, ',', iter.pos) as name
4   from (select id as pos from t10) iter,
5        (select cast(name as text) as name from v) src
7  where iter.pos <=
8         length(src.name)-length(replace(src.name, ',', ''))+1
9         ) x
10  where pos = 2

```

SQL Server

Функция `STRING_SPLIT` выполняет всю работу по разбиению строк на составляющие подстроки, но может обрабатывать только одну строку таблицы зараз. Поэтому для предоставления ей данных в требуемом формате используем функцию `STRING_AGG` в обобщенном табличном выражении:

```

1 with agg_tab(name)
2   as
3   (select STRING_AGG(name, ',') from V)
4 select value from
5   STRING_SPLIT(
6   (select name from agg_tab), ',')

```

Обсуждение

DB2

Во вложенном запросе выполняем обход строк, чтобы разбить их на составляющие значения, и сохраняем результат в представлении X:

```
select ', ' || csv.name || ', ' as name,
       iter.pos
       from v csv,
       (select row_number() over() pos from t100 ) iter
       where iter.pos <= length(csv.name)+2
```

EMPS	POS
,tina,gina,jaunita,regina,leena,	1
,tina,gina,jaunita,regina,leena,	2
,tina,gina,jaunita,regina,leena,	3
...	

Затем извлекаем отдельные значения каждой строки:

```
select pos, name, substr(name, pos) c,
       row_number() over(partition by name
                          order by length(substr(name, pos)) desc) rn
       from (
select ', ' || csv.name || ', ' as name,
       cast(iter.pos as integer) as pos
       from v csv,
       (select row_number() over() pos from t100 ) iter
       where iter.pos <= length(csv.name)+2
       ) x
       where length(substr(name,pos)) > 1
```

POS	EMPS	C	RN
1	,mo,larry,curly,	,mo,larry,curly,	1
2	,mo,larry,curly,	mo,larry,curly,	2
3	,mo,larry,curly,	o,larry,curly,	3
4	,mo,larry,curly,	,larry,curly,	4

Получив таким образом доступ к отдельным частям строки, просто определяем строки, которые хотим оставить. Это строки, которые начинаются с запятой, а остальные строки можно отбросить:

```
select pos, name, substr(name,pos) c,
       row_number() over(partition by name
                          order by length(substr(name, pos)) desc) rn
       from (
select ', ' || csv.name || ', ' as name,
       cast(iter.pos as integer) as pos
       from v csv,
       (select row_number() over() pos from t100 ) iter
```

```

where iter.pos <= length(csv.name)+2
    ) x
where length(substr(name,pos)) > 1
    and substr(substr(name,pos),1,1) = ','

```

POS	EMPS	C	RN
1	,mo,larry,curly,	,mo,larry,curly,	1
4	,mo,larry,curly,	,larry,curly,	2
10	,mo,larry,curly,	,curly,	3
1	,tina,gina,jaunita,regina,leena,	,tina,gina,jaunita,regina,leena,	1
6	,tina,gina,jaunita,regina,leena,	,gina,jaunita,regina,leena,	2
11	,tina,gina,jaunita,regina,leena,	,jaunita,regina,leena,	3
19	,tina,gina,jaunita,regina,leena,	,regina,leena,	4
26	,tina,gina,jaunita,regina,leena,	,leena,	5

Это важный шаг, т. к. в нем осуществляется подготовка к извлечению n -й подстроки. Обратите внимание, что в этот запрос не попали многие строки вследствие наличия в предикате WHERE следующего условия:

```
substr(substr(name,pos),1,1) = ','
```

Отметим, что строка `,mo,larry,curly,` ранее имела ранг 4, но теперь у нее ранг 2. Не забываем, что предикат WHERE выполняется прежде SELECT, поэтому оставляются строки с запятыми, и только *после* этого функция ROW_NUMBER выполняет ранжирование строк. На этом этапе можно легко видеть, что для того, чтобы получить n -ю подстроку, нам требуются строки, для которых RN равно n . Последний шаг — оставить только требуемые строки (в нашем случае это строки, для которых значение RN равно 2) и посредством функции SUBSTR извлечь из этой строки имя. Требуемое имя — первое в строке: `larry` в строке `,larry,curly,` и `gina` в строке `,gina,jaunita,regina,leena,.`

MySQL

Вложенный запрос X выполняет обход всех строк. Количество значений в каждой строке определяется по количеству содержащихся в ней разделителей:

```

select iter.pos, src.name
  from (select id pos from t10) iter,
       V src
 where iter.pos <=
       length(src.name)-length(replace(src.name,',',''))

```

pos	name
1	mo,larry,curly
2	mo,larry,curly
1	tina,gina,jaunita,regina,leena
2	tina,gina,jaunita,regina,leena

```
| 3 | tina,gina,jaunita,regina,leena |
| 4 | tina,gina,jaunita,regina,leena |
+-----+-----+
```

В нашем случае количество строк на одну меньше, чем количество значений в каждой строке, т. к. это все, что требуется. Парсинг для выбора требуемых значений обеспечивает функция `SUBSTRING_INDEX`:

```
select iter.pos,src.name name1,
       substring_index(src.name,',',iter.pos) name2,
       substring_index(
         substring_index(src.name,',',iter.pos),',',-1) name3
  from (select id pos from t10) iter,
       V src
 where iter.pos <=
        length(src.name)-length(replace(src.name,',',''))
```

```
+-----+-----+-----+-----+
| pos | name1                                | name2                                | name3 |
+-----+-----+-----+-----+
| 1   | mo,larry,curly                       | mo                                   | mo    |
| 2   | mo,larry,curly                       | mo,larry                             | larry |
| 1   | tina,gina,jaunita,regina,leena       | tina                                  | tina  |
| 2   | tina,gina,jaunita,regina,leena       | tina,gina                            | gina  |
| 3   | tina,gina,jaunita,regina,leena       | tina,gina,jaunita                   | jaunita |
| 4   | tina,gina,jaunita,regina,leena       | tina,gina,jaunita,regina            | regina |
+-----+-----+-----+-----+
```

Чтобы облегчить понимание работы вложенных вызовов функции `SUBSTRING_INDEX`, мы показали дополнительные три поля имен. Вложенный вызов возвращает все символы слева от *n*-й запятой, а внешний — все символы справа от первой обнаруженной им запятой, начиная с конца строки. В завершение выбирается значение, для которого POS равно *n*, в нашем случае 2, и помещается в столбец `NAME3`.

SQL Server

Основную работу в приведенном решении выполняет функция `STRING_SPLIT`, но она должна получить должным образом сформированные данные. Обобщенное табличное выражение просто преобразовывает две строки столбца `V.names` в одно значение, что требуется для функции `STRING_SPLIT`, возвращающей таблицу.

Oracle

Вложенный запрос выполняет обход всех строк. Количество возвращений каждой строки определяется количеством значений в ней. Решение определяет количество значений в каждой строке, подсчитывая количество содержащихся в ней разделителей (запятых в нашем случае). Поскольку строки заключены в запяты, то количество значений в строке будет на одно меньше, чем количество запятых. Строки

затем объединяются оператором UNION и добавляются в таблицу, кардинальность которой равна как минимум количеству значений в самой длинной строке. Функции SUBSTR и INSTR выполняют парсинг каждой строки, используя значение POS:

```
select iter.pos, src.name,
       substr( src.name,
              instr( src.name, ',', 1, iter.pos )+1,
              instr( src.name, ',', 1, iter.pos+1 )
              - instr( src.name, ',', 1, iter.pos )-1) sub
  from (select ' ,||name||', ' as name from v) src,
       (select rownum pos from emp) iter
 where iter.pos < length(src.name)-length(replace(src.name, ','))
```

POS	NAME	SUB
1	,mo,larry,curly,	mo
1	, tina,gina,jaunita,regina,leena, tina	tina
2	,mo,larry,curly,	larry
2	, tina,gina,jaunita,regina,leena, gina	gina
3	,mo,larry,curly,	curly
3	, tina,gina,jaunita,regina,leena, jaunita	jaunita
4	, tina,gina,jaunita,regina,leena, regina	regina
5	, tina,gina,jaunita,regina,leena, leena	leena

В первом вызове функции INSTR в функции SUBSTR определяется начальная позиция подстроки, которую нужно извлечь. В следующем вызове INSTR в SUBSTR определяется позиция n -й запятой (она же и начальная позиция), а также позиция $n+1$ запятой. Длина извлекаемой строки определяется, вычитая первое значение из второго. Поскольку при парсинге каждое значение помещается в отдельную строку таблицы, чтобы оставить n -ю подстроку, просто задаем WHERE POS = n (в нашем случае, где POS = 2, оставляется вторая подстрока в списке).

postgresql

Вложенный запрос X выполняет обход всех строк. Количество возвращаемых строк определяется количеством значений в исходной строке. Количество значений будет равно количеству разделителей (запятых) плюс 1. Функция SPLIT_PART находит n -й разделитель по значениям POS и выполняет парсинг строки для разбивки ее на отдельные значения:

```
select iter.pos, src.name as name1,
       split_part(src.name, ',', iter.pos) as name2
  from (select id as pos from t10) iter,
       (select cast(name as text) as name from v) src
 where iter.pos <=
       length(src.name)-length(replace(src.name, ','))
```

pos	name1	name2
1	mo,larry,curly	mo
2	mo,larry,curly	larry
3	mo,larry,curly	curly
1	tina,gina,jaunita,regina,leena	tina
2	tina,gina,jaunita,regina,leena	gina
3	tina,gina,jaunita,regina,leena	jaunita
4	tina,gina,jaunita,regina,leena	regina
5	tina,gina,jaunita,regina,leena	leena

Два столбца имен NAME приводятся для того, чтобы показать, как функция `SPLIT_PART` выполняет парсинг строк для разбивки их на отдельные значения, используя значение `POS`. После разбивки всех строк на отдельные значения выполняется последняя операция — выбор из полученных строк тех, для которых значение `POS` соответствует требуемой *n*-й подстроке, — в нашем случае 2.

6.15. Парсинг IP-адреса

ЗАДАЧА

Требуется выполнить парсинг IP-адреса и поместить значения полей в отдельные столбцы таблицы. Возьмем, например, такой IP-адрес:

```
111.22.3.4
```

Мы хотим получить из него следующий результат выполнения запроса:

A	B	C	D
111	22	3	4

РЕШЕНИЕ

Подробности решения определяются встроенными функциями используемой СУБД. Но, независимо от конкретной СУБД, алгоритм решения состоит в нахождении точек и выборе смежных с ними чисел.

DB2

Эмулируем цикл перебора IP-адреса посредством рекурсивного блока `WITH`, извлекая группы значений с помощью функции `SUBSTR`. В начало строки IP-адреса добавляется точка, чтобы все группы чисел были одинаковыми, — это позволяет обрабатывать их одним и тем же способом:

```
1 with x (pos,ip) as (
2   values (1, '.92.111.0.222')
3   union all
4   select pos+1,ip from x where pos+1 <= 20
5 )
```



```

6 select max(case when rn=1 then e end) a,
7         max(case when rn=2 then e end) b,
8         max(case when rn=3 then e end) c,
9         max(case when rn=4 then e end) d
10 from (
11 select pos,c,d,
12        case when posstr(d, '.') > 0 then substr(d,1,posstr(d, '.')-1)
13           else d
14        end as e,
15        row_number() over( order by pos desc) rn
16 from (
17 select pos, ip,right(ip,pos) as c, substr(right(ip,pos),2) as d
18 from x
19 where pos <= length(ip)
20 and substr(right(ip,pos),1,1) = '.'
21 ) x
22 ) y

```

MySQL

Для парсинга и разбивки IP-адреса на группы чисел используем функцию SUBSTR_INDEX:

```

1 select substring_index(substring_index(y.ip, '.',1), '.',-1) a,
2        substring_index(substring_index(y.ip, '.',2), '.',-1) b,
3        substring_index(substring_index(y.ip, '.',3), '.',-1) c,
4        substring_index(substring_index(y.ip, '.',4), '.',-1) d
5 from (select '92.111.0.2' as ip from t1) y

```

Oracle

Для парсинга и разбивки IP-адреса на группы чисел используем встроенные функции SUBSTR и INSTR:

```

1 select ip,
2        substr(ip, 1, instr(ip, '.')-1 ) a,
3        substr(ip, instr(ip, '.')+1,
4               instr(ip, '.',1,2)-instr(ip, '.')-1 ) b,
5        substr(ip, instr(ip, '.',1,2)+1,
6               instr(ip, '.',1,3)-instr(ip, '.',1,2)-1 ) c,
7        substr(ip, instr(ip, '.',1,3)+1 ) d
8 from (select '92.111.0.2' as ip from t1)

```

PostgreSQL

Для парсинга и разбивки IP-адреса на группы чисел используем встроенную функцию SPLIT_PART:

```

1 select split_part(y.ip, '.',1) as a,
2        split_part(y.ip, '.',2) as b,

```

```

3     split_part(y.ip, '.', 3) as c,
4     split_part(y.ip, '.', 4) as d
5   from (select cast('92.111.0.2' as text) as ip from t1) as y

```

SQL Server

Эмулируем цикл перебора IP-адреса посредством рекурсивного блока WITH, извлекая группы значений с помощью функции SUBSTR. В начало строки IP-адреса добавляется точка, чтобы все группы чисел были одинаковыми, — это позволяет обрабатывать их одним и тем же способом:

```

1 with x (pos,ip) as (
2   select 1 as pos, '.92.111.0.222' as ip from t1
3   union all
4   select pos+1, ip from x where pos+1 <= 20
5 )
6 select max(case when rn=1 then e end) a,
7        max(case when rn=2 then e end) b,
8        max(case when rn=3 then e end) c,
9        max(case when rn=4 then e end) d
10   from (
11 select pos, c, d,
12        case when charindex('.', d) > 0
13           then substring(d, 1, charindex('.', d)-1)
14           else d
15        end as e,
16        row_number() over(order by pos desc) rn
17   from (
18 select pos, ip, right(ip, pos) as c,
19        substring(right(ip, pos), 2, len(ip)) as d
20   from x
21  where pos <= len(ip)
22  and substring(right(ip, pos), 1, 1) = '.'
23   ) x
24   ) y

```

Обсуждение

Задачу перебора и извлечения составных частей строки можно легко решить с помощью встроенных функций используемой СУБД. Особенность решения состоит в нахождении символов точки в адресе, после чего можно извлечь числа между ними.

В рецепте 6.17 рассматривается возможность использования в большинстве СУБД регулярных выражений. Этот подход также можно применить и для извлечения групп чисел из IP-адреса.

6.16. Сравнение строк по их звучанию

ЗАДАЧА

Кроме ошибок правописания и разных вариантов правильного написания слов — например, британских и американских вариантов — существует много ситуаций, когда слова, которые нужно сравнить, пишутся по-разному. К счастью, язык SQL поддерживает средство для представления звучания слов, что позволяет сравнивать строки по их звучанию, даже если они пишутся по-разному.

Предположим, что имеется список фамилий авторов, причем некоторые записаны в более ранние времена, когда правила правописания еще не были полностью установлены, а в нескольких из фамилий допущены орфографические или типографские ошибки. Далее приводится пример такого списка:

```
a_name
----
1 Johnson
2 Jonson
3 Jonsen
4 Jensen
5 Johnsen
6 Shakespeare
7 Shakspear
8 Shaekspir
9 Shakespar
```

Это, конечно, лишь часть более обширного списка вариантов написания этих имен, но нам его вполне достаточно, чтобы попытаться найти в нем потенциально фонетически совпадающие имена. И хотя в этом примере возможны несколько решений, одно из этих решений может выглядеть так (значение последнего столбца станет понятным к концу описания этого рецепта):

a_name1	a_name2	soundex_name
----	----	----
Jensen	Johnson	J525
Jensen	Jonson	J525
Jensen	Jonsen	J525
Jensen	Johnsen	J525
Johnsen	Johnson	J525
Johnsen	Jonson	J525
Johnsen	Jonsen	J525
Johnsen	Jensen	J525
...		
Jonson	Jensen	J525
Jonson	Johnsen	J525
Shaekspir	Shakspear	S216
Shakespar	Shakespeare	S221
Shakespeare	Shakespar	S221
Shakspear	Shaekspir	S216

РЕШЕНИЕ

Используя функцию `SOUNDEX`, преобразовываем строки символов в представления их звучания на английском языке. Затем сравниваем значения столбца, используя простое самообъединение:

```
1 select an1.a_name as name1, an2.a_name as name2,
2 SOUNDEX(an1.a_name) as Soudex_Name
3 from author_names an1
4 join author_names an2
5 on (SOUNDEX(an1.a_name)=SOUNDEX(an2.a_name)
6 and an1.a_name not like an2.a_name)
```

Обсуждение

Идея, лежащая в основе функции `SOUNDEX`, возникла еще на заре развития баз данных и компьютерных вычислений — когда Бюро переписи США предприняло попытку решить проблему разного написания имен собственных людей и географических объектов. Существует много алгоритмов, пытающихся решить ту же задачу, что и функция `SOUNDEX`, и, конечно же, имеются альтернативные версии этой функции для других языков, кроме английского. Но в этой книге мы рассматриваем функцию `SOUNDEX`, которая поддерживается большинством СУБД.

Функция `SOUNDEX` сохраняет первую букву имени и заменяет остальные числами. При этом буквы, имеющие сходное звучание, заменяются одинаковыми числами. Например, буквы *m* и *n* обе заменяются числом 5.

В приведенном примере возвращенные функцией `SOUNDEX` значения показаны в столбце `soudex_name`. Это сделано здесь просто с целью показать, что происходит в процессе решения, и наличие этого столбца не обязательно для решения. Некоторые СУБД даже оснащены функцией, которая скрывает результат выполнения функции `SOUNDEX`. Например, функция `DIFFERENCE` СУБД SQL Server сравнивает посредством `SOUNDEX` две строки и возвращает коэффициент подобия в диапазоне от 0 до 4. Значение 4 при этом означает идеальное совпадение выходных значений `SOUNDEX` для двух этих строк.

Иногда функции `SOUNDEX` вполне хватает для решения поставленной задачи, а иногда нет. В книге «Data Matching» (автор Peter Christen, 2012 г.) можно найти другие отвечающие вашим требованиям и желаниям алгоритмы, которые часто (но не всегда) достаточно легко поддаются реализации в виде определяемой пользователем функции или на другом языке программирования.

6.17. Обнаружение текста, не совпадающего с шаблоном

ЗАДАЧА

Требуется найти текстовое поле, содержащее определенные структурированные текстовые значения (например, телефонные номера), а также случаи таких значе-

ний с неправильным структурированием. Например, у нас есть следующие исходные данные:

```
select emp_id, text
       from employee_comment
```

EMP_ID	TEXT
7369	126 Varnum, Edmore MI 48829, 989 313-5351
7499	1105 McConnell Court Cedar Lake MI 48812 Home: 989-387-4321 Cell: (237) 438-3333

В этих данных нам нужно найти и вывести строки, содержащие телефонные номера в неправильном формате. Например, вывести следующую строку, поскольку в ней используются два разных разделителя для групп цифр телефонного номера:

```
7369 126 Varnum, Edmore MI 48829, 989 313-5351
```

Правильными считаются только такие телефонные номера, в которых используется одинаковый символ для обоих разделителей.

РЕШЕНИЕ

Решение этой задачи состоит из нескольких частей:

1. Определяем способ описания совокупности возможных телефонных номеров, которые нужно включить в рассмотрение.
2. Удаляем из рассмотрения все телефонные номера, отформатированные должным образом.
3. Проверяем, остались ли в списке предположительные телефонные номера. Если остались, то эти номера отформатированы неправильно:

```
select emp_id, text
       from employee_comment
 where regexp_like(text, '[0-9]{3}[-. ] [0-9]{3}[-. ] [0-9]{4}')
 and regexp_like(
       regexp_replace(text,
       '[0-9]{3}([-. ] [0-9]{3})\1[0-9]{4}', '***'),
       '[0-9]{3}[-. ] [0-9]{3}[-. ] [0-9]{4}')
```

EMP_ID	TEXT
7369	126 Varnum, Edmore MI 48829, 989 313-5351
7844	989-387.5359
9999	906-387-1698, 313-535.8886

Каждая из этих строк содержит по крайней мере один предположительно неправильно отформатированный телефонный номер.

Обсуждение

Суть этого решения состоит в определении «предположительных телефонных номеров». С учетом того, что телефонные номера хранятся в поле комментария (`comment`), любой текст в этом поле можно рассматривать как неправильно отформатированный телефонный номер. Однако нам нужно сузить поле рассмотрения к более разумному набору значений, поскольку мы не хотим, чтобы результат содержал, например, следующий текст:

```
EMP_ID TEXT
```

```
-----
7900 Cares for 100-year-old aunt during the day. Schedule only
for evening and night shifts.
```

Очевидно, что в этой строке нет никаких телефонных номеров, не говоря уже о неправильно отформатированных. Очевидно для нас, людей? Но вопрос в том, как нам заставить СУБД «увидеть» это? Мы полагаем, что вам понравится ответ. Читайте далее.



Этот рецепт взят (с разрешения) из статьи «Regular Expression Anti-Patterns» («Антишаблоны регулярных выражений»), автор Jonathan Gennick.

В решении используется шаблон `Pattern A` для определения набора «предположительных» телефонных номеров, которые нужно рассмотреть:

```
Pattern A: [0-9]{3}[-. ]{0-9}{3}[-. ]{0-9}{4}
```

Шаблон `Pattern A` выполняет проверку на наличие двух групп из трех цифр, за которыми следует одна группа из четырех цифр. Разделителем между группами может служить тире (-), точка (.) или пробел. При желании можно создать более сложный шаблон. Например, можно также рассматривать телефонные номера из семи цифр. Но не стоит отвлекаться на несущественные моменты. Суть в том, что нам нужно как-то определить для рассмотрения совокупность возможных строк телефонных номеров. Примем для нашей задачи, что эта совокупность определяется шаблоном `Pattern A`. Можно определить какой-либо другой шаблон `Pattern A`, но общее решение останется применимым.

В решении `Pattern A` используется в предикате `WHERE`, чтобы обеспечить рассмотрение только строк, содержащих потенциальные (согласно определению шаблона) телефонные номера:

```
select emp_id, text
       from employee_comment
       where regexp_like(text, '[0-9]{3}[-. ]{0-9}{3}[-. ]{0-9}{4}')
```

Далее мы определяем, как должен выглядеть «правильный» телефонный номер. В решении это делается с помощью шаблона `Pattern B`:

```
Pattern B: [0-9]{3}([-. ]){0-9}{3}\1{0-9}{4}
```

В этом шаблоне первое подвыражение обозначается символами \1. Любой символ, совпадающий с ([-.]), также должен совпадать с \1. Шаблон `Pattern B` описывает «правильно» отформатированные телефонные номера, которые необходимо устранить из рассмотрения (т. к. они не являются «неправильными»). В решении правильно отформатированные телефонные номера удаляются вызовом функции `REGEXP_REPLACE`:

```
regexp_replace(text,
               '[0-9]{3}([-. ])[0-9]{3}\1[0-9]{4}', '***'),
```

Этот вызов функции `REGEXP_REPLACE` происходит в предикате `WHERE`. В результате все правильно отформатированные телефонные номера преобразовываются в строки из трех звездочек (**). Опять же, `Pattern B` может быть любым — главное, что он содер­жал описание требуемого нам шаблона.

После замены правильно отформатированных телефонных номеров строками из трех звездочек все оставшиеся «возможные» телефонные номера по определению должны быть неправильно отформатированными. В решении проверка на наличие неправильно отформатированных телефонных номеров осуществляется обработкой вывода первого вызова `REGEXP_LIKE` другим вызовом `REGEXP_LIKE`:

```
and regexp_like(
    regexp_replace(text,
                  '[0-9]{3}([-. ])[0-9]{3}\1[0-9]{4}', '***'),
    '[0-9]{3}[-. ]{0-9}{3}[-. ]{0-9}{4}')
```



Регулярные выражения — сама по себе обширная тема, требующая практики, чтобы научиться работать с ними. Но, овладев работой с регулярными выражениями, вы сможете использовать их для определения обширного круга шаблонов строк. В качестве одного из средств для повышения своего уровня знаний по части регулярных выражений мы можем порекомендовать книгу «Mastering Regular Expression», автор Jeffrey Friedl.

6.18. Подведем итоги

Сопоставление строк может быть трудоемкой задачей. В последние версии SQL был добавлен круг средств, овладев которыми вы сможете уменьшить эту для себя трудоемкость. Хотя встроенные функции SQL для работы со строками позволяют выполнять множество задач, использование регулярных выражений, поддержка которых современными СУБД постоянно улучшается, позволяет делать это совсем на другом уровне.

Операции с числами

В этой главе рассматриваются операции над числами, включая различные вычисления. Хотя SQL — это не наилучшее средство для сложных вычислений, повседневные рутинные вычисления он обеспечивает достаточно эффективно.

Кроме того, базы данных, поддерживающие SQL, скорее всего, продолжают оставаться наиболее распространенным средством для хранения, поиска и организации данных, поэтому использование SQL для анализа и обработки этих данных имеет важное значение для любого, кто использует эти данные. Рассматриваемые в этой главе методы помогут специалистам по работе с данными решить, какие данные являются наиболее перспективными для дальнейшего анализа.



В некоторых рецептах этой главы используются агрегатные функции и оператор `GROUP BY`. Если вы не знакомы с принципами группирования, прочитайте хотя бы первый основной раздел «Группировка» приложения 1.

7.1. Вычисление среднего

ЗАДАЧА

Требуется определить среднее значение столбца либо для всех строк, либо только для определенного подмножества строк. Например, вычислить среднюю заработную плату как для всех служащих, так и для каждого отдела.

РЕШЕНИЕ

Среднее значение зарплаты для всех служащих вычисляется простым применением функции `AVG` к столбцу зарплат. При этом для вычисления среднего для всех служащих предикат `WHERE` не используется:

```
1 select avg(sal) as avg_sal
2   from emp
```

```
      AVG_SAL
-----
2073.21429
```

А чтобы вычислить среднее значение для каждого отдела, создаем для каждого отдела группу, используя для этого оператор `GROUP BY`:


```

1 select deptno, avg(sal) as avg_sal
2   from emp
3  group by deptno

```

DEPTNO	AVG_SAL
10	2916.66667
20	2175
30	1566.66667

Обсуждение

Чтобы вычислить среднее, когда группой или окном является вся таблица, просто применяем функцию `AVG` к требуемому столбцу, не используя оператор `GROUP BY`. Важно отметить, что функция `AVG` игнорирует значения `NULL`. Далее приводится пример эффекта игнорирования значений `NULL`:

```

create table t2(sal integer)
insert into t2 values (10)
insert into t2 values (20)
insert into t2 values (null)
select avg(sal) select distinct 30/2
   from t2 from t2

```

AVG(SAL)	30/2
15	15

```

select avg(coalesce(sal,0)) select distinct 30/3
   from t2 from t2

```

AVG(COALESCE(SAL,0))	30/3
10	10

Функция `COALESCE` возвращает первое значение не-`NULL` в списке переданных ей значений. Если значения `NULL` столбца `SAL` преобразовать в число 0, то среднее меняется. Поэтому при использовании агрегатных функций всегда следует учитывать, каким образом надо обрабатывать значения `NULL`.

Во второй части решения записи служащих группируются по отделам с использованием для этого оператора `GROUP BY` (строка 3). Оператор `GROUP BY` автоматически вызывает выполнение агрегатных функций, таких как `AVG`, и возвращает результат для каждой группы. В приведенном примере функция `AVG` выполняется для каждого подмножества записей, сгруппированных по номеру отдела.

Кстати, включать в списке оператора `SELECT` столбец, по которому выполняется группировка, не обязательно:

```

select avg(sal)
   from emp
  group by deptno

```

```

AVG(SAL)
-----
2916.66667
      2175
1566.66667

```

Хотя здесь столбец DEPTNO в списке оператора SELECT не указывается, группирование по номерам отделов все равно осуществляется. Указание столбца для группирования в списке оператора SELECT часто улучшает читаемость кода запроса, но не является обязательным. Однако если столбцы для группирования указаны в списке SELECT, они обязательно должны быть также указаны и в операторе GROUP BY.

См. также

В приложении 1 представлен краткий обзор функциональных возможностей оператора GROUP BY.

7.2. Определение минимального и/или максимального значения столбца

ЗАДАЧА

Требуется определить минимальное и максимальное значения заданного столбца. Например, вычислить наименьшую и наибольшую заработную плату как для всех служащих, так и для каждого отдела.

РЕШЕНИЕ

Для поиска наименьшего и наибольшего значений столбца используются функции MIN и MAX, соответственно:

```

1 select min(sal) as min_sal, max(sal) as max_sal
2    from emp

```

```

  MIN_SAL    MAX_SAL
-----
      800      5000

```

А для поиска наименьшего и наибольшего значений зарплаты для каждого отдела совместно с функциями MIN и MAX используется оператор GROUP BY:

```

1 select deptno, min(sal) as min_sal, max(sal) as max_sal
2    from emp
3   group by deptno

```

```

  DEPTNO    MIN_SAL    MAX_SAL
-----
      10         1300         5000
      20          800         3000
      30          950         2850

```

Обсуждение

При поиске наибольшего или наименьшего значения, когда группой или окном является вся таблица, просто применяем к требуемому столбцу функцию `MIN` или `MAX` без оператора `GROUP BY`.

Следует иметь в виду, что функции `MIN` и `MAX` игнорируют значения `NULL` и что группы могут как содержать отдельные значения `NULL`, так и полностью состоять из значений `NULL`. Далее приводится несколько примеров, в последнем из которых запрос с использованием `GROUP BY` возвращает значения `NULL` для двух групп (отделы `DEPTNO 10` и `20`):

```
select deptno, comm
       from emp
      where deptno in (10,30)
     order by 1
```

DEPTNO	COMM
10	
10	
10	
30	300
30	500
30	
30	0
30	1300
30	

```
select min(comm), max(comm)
       from emp
```

MIN (COMM)	MAX (COMM)
0	1300

```
select deptno, min(comm), max(comm)
       from emp
      group by deptno
```

DEPTNO	MIN (COMM)	MAX (COMM)
10		
20		
30	0	1300

Как отмечено в *приложении 1*, даже если в списке оператора `SELECT` не указано никаких столбцов, а имеется только агрегатная функция, группирование можно все равно осуществить по другим столбцам таблицы:

```
select min(comm) , max(comm)
       from emp
       group by deptno
```

```
MIN (COMM)  MAX (COMM)
-----
           0           1300
```

Хотя здесь столбец DEPTNO в списке оператора SELECT не указывается, группирование по номерам отделов все равно осуществляется. Указание столбца для группирования в списке оператора SELECT часто улучшает читаемость кода запроса, но не является обязательным. Однако все столбцы для группирования в списке SELECT обязательно должны быть указаны также и в операторе GROUP BY.

См. также

В приложении 1 приводится краткий обзор функциональных возможностей GROUP BY.

7.3. Суммирование значений столбца

ЗАДАЧА

Требуется вычислить сумму всех значений столбца — например, общую заработную плат всех служащих.

РЕШЕНИЕ

Чтобы вычислить общую сумму, когда группой или окном является вся таблица, просто применяем функцию SUM к требуемому столбцу, не используя оператора GROUP BY:

```
1 select sum(sal)
2   from emp
```

```
SUM (SAL)
-----
    29025
```

Для вычисления подсумм нескольких групп или окон данных функция SUM используется с оператором GROUP BY. В следующем примере вычисляются подсуммы зарплат по отделу:

```
1 select deptno, sum(sal) as total_for_dept
2   from emp
3  group by deptno
```

```
DEPTNO TOTAL_FOR_DEPT
-----
      10           8750
      20          10875
      30           9400
```

Обсуждение

При вычислении подсумм зарплат для каждого отдела создаются группы или «окна» данных. Заработная плата всех служащих каждого отдела складывается вместе, создавая общую сумму соответствующего отдела. Это пример работы агрегатной операции SQL, т. к. целью является не подробная информация, как, например, зарплата каждого отдельного служащего, а общая сумма для всего отдела. Важно иметь в виду, что функция `SUM` игнорирует значения `NULL`, но допускается существование групп, содержащих только значения `NULL`, как показано в следующих примерах. Никто из служащих отдела 10 не получает комиссионных, поэтому операция `SUM` по столбцу `COMM` для этого отдела выполняется по группе значений `NULL` и ожидаемо возвращает значение `NULL`:

```
select deptno, comm
  from emp
 where deptno in (10,30)
 order by 1
```

DEPTNO	COMM
10	
10	
10	
30	300
30	500
30	
30	0
30	1300
30	

```
select sum(comm)
  from emp
```

SUM(COMM)
2100

```
select deptno, sum(comm)
  from emp
 where deptno in (10,30)
 group by deptno
```

DEPTNO	SUM(COMM)
10	
30	2100

См. также

В приложении 1 приводится краткий обзор функциональных возможностей оператора `GROUP BY`.

7.4. Подсчет строк в таблице

ЗАДАЧА

Требуется подсчитать количество строк в таблице или количество значений в столбце таблицы — например, вычислить общее количество служащих, а также количество служащих в каждом отделе.

РЕШЕНИЕ

Когда группой или окном является вся таблица, для подсчета строк используем функцию `COUNT` с символом `*`:

```
1 select count(*)
2   from emp
```

```
   COUNT (*)
-----
          14
```

А для подсчета строк в нескольких группах или окнах данных функция `COUNT` используется с оператором `GROUP BY`:

```
1 select deptno, count(*)
2   from emp
3  group by deptno
```

```
   DEPTNO   COUNT (*)
-----
         10           3
         20           5
         30           6
```

Обсуждение

При подсчете строк для каждого отдела создаются группы или «окна» данных. С каждой новой строкой служащего счет инкрементируется на единицу, образуя в конечном итоге общее количество для соответствующего отдела. Это пример работы агрегатной операции SQL, т. к. целью является не подробная информация, как, например, зарплата или должность каждого отдельного служащего, а конечный результат для каждого отдела. Важно иметь в виду, что функция `COUNT` игнорирует значения `NULL` при передаче ей в виде аргумента столбца, но учитывает эти значения, когда ей передается символ `*` или любая константа. Рассмотрим следующие запросы:

```
select deptno, comm
   from emp
```

DEPTNO	COMM
20	
30	300
30	500
20	
30	1300
30	
10	
20	
10	
30	0
20	
30	
20	
10	

```
select count(*), count(deptno), count(comm), count('hello')
from emp
```

COUNT(*)	COUNT(DEPTNO)	COUNT(COMM)	COUNT('HELLO')
14	14	4	14

```
select deptno, count(*), count(comm), count('hello')
from emp
group by deptno
```

DEPTNO	COUNT(*)	COUNT(COMM)	COUNT('HELLO')
10	3	0	3
20	5	0	5
30	6	4	6

Если все строки столбца, переданного функции COUNT, содержат значения NULL или если таблица пустая, функция возвращает значение 0. Нужно заметить, что, даже если в списке оператора SELECT не указано никаких столбцов, а имеется только агрегатная функция, группирование можно все равно осуществить по другим столбцам таблицы:

```
select count(*)
from emp
group by deptno
```

COUNT(*)
3
5
6

Обратите внимание, что, хотя в приведенном примере столбец `DEPTNO` в списке оператора `SELECT` не указывается, группирование по номерам отделов все равно осуществляется. Указание столбца для группирования в списке оператора `SELECT` часто улучшает читаемость кода запроса, но не является обязательным. Однако при указании столбца в списке оператора `SELECT` его также обязательно нужно указать в операторе `GROUP BY`.

См. также

В *приложении 1* приводится краткий обзор функциональных возможностей оператора `GROUP BY`.

7.5. Подсчет значений столбца

ЗАДАЧА

Требуется подсчитать количество не-NULL значений в столбце — например, узнать количество служащих, работающих на комиссионной основе.

РЕШЕНИЕ

Применяем функцию `COUNT`, передавая ей в качестве аргумента столбец `COMM` таблицы `EMP`:

```
select count(comm)
from emp
```

```
COUNT (COMM)
-----
4
```

Обсуждение

При передаче функции `COUNT` звездочки в качестве аргумента (`COUNT(*)` — как в предыдущем рецепте) мы в действительности подсчитываем количество строк столбца, независимо от его значений. Поэтому в расчет берутся столбцы, содержащие как определенные значения, так и значения `NULL`. Но при передаче в качестве аргумента названия столбца подсчитываются только определенные (т. е. не-NULL) значения столбца. Этот момент уже упоминался в *разд. «Обсуждение»* предыдущего рецепта. В приведенном здесь решении выражение `COUNT (COMM)` возвращает количество значений не-NULL в столбце `COMM`. Поскольку комиссионные данные имеют только служащие, работающие на комиссионной основе, то вызов `COUNT (COMM)` возвращает количество таких служащих.

7.6. Вычисление текущей суммы

ЗАДАЧА

Требуется вычислять текущую сумму значений столбца.

РЕШЕНИЕ

Далее приводится пример решения, демонстрирующий вычисление текущей суммы заработной платы для всех служащих. Для удобочитаемости результаты упорядочены по столбцу SAL, что позволяет с легкостью наблюдать процесс вычисления текущей суммы:

```
1 select ename, sal,
2         sum(sal) over (order by sal,empno) as running_total
3   from emp
4  order by 2
```

ENAME	SAL	RUNNING_TOTAL
SMITH	800	800
JAMES	950	1750
ADAMS	1100	2850
WARD	1250	4100
MARTIN	1250	5350
MILLER	1300	6650
TURNER	1500	8150
ALLEN	1600	9750
CLARK	2450	12200
BLAKE	2850	15050
JONES	2975	18025
SCOTT	3000	21025
FORD	3000	24025
KING	5000	29025

Обсуждение

Задача вычисления текущей суммы с легкостью решается посредством оконной функции SUM OVER. В приведенном решении в операторе ORDER BY указывается не только столбец SAL, но также и столбец первичного ключа EMPNO, чтобы не учитывать в текущей сумме дубликатов значений. Возможная проблема с дубликатами, которая может возникнуть без указания столбца первичного ключа, показана в столбце RUNNING_TOTAL2 в следующем примере:

```
select empno, sal,
       sum(sal)over(order by sal,empno) as running_total1,
       sum(sal)over(order by sal) as running_total2
  from emp
 order by 2
```

ENAME	SAL	RUNNING_TOTAL1	RUNNING_TOTAL2
SMITH	800	800	800
JAMES	950	1750	1750
ADAMS	1100	2850	2850
WARD	1250	4100	5350
MARTIN	1250	5350	5350
MILLER	1300	6650	6650
TURNER	1500	8150	8150
ALLEN	1600	9750	9750
CLARK	2450	12200	12200
BLAKE	2850	15050	15050
JONES	2975	18025	18025
SCOTT	3000	21025	24025
FORD	3000	24025	24025
KING	5000	29025	29025

Значения в столбце `RUNNING_TOTAL2` для служащих `WARD`, `MARTIN`, `SCOTT` и `FORD` — неправильные. Значения их заработных плат встречаются несколько раз, и эти дубликаты складываются, а затем добавляются к текущей сумме. Вот поэтому для получения правильных результатов, которые отображаются в столбце `RUNNING_TOTAL1`, требуется указывать столбец `EMPNO`, все значения которого являются уникальными. Рассмотрим следующий случай: для служащего `ADAMS` значения `RUNNING_TOTAL1` и `RUNNING_TOTAL2` одинаковы — 2850. Для служащего `WARD` добавляем к 2850 его зарплату 1250 и должны получить 4100. Но значение `RUNNING_TOTAL2` равно 5350. Почему? Так как значения `SAL` для служащих `WARD` и `MARTIN` одинаковые, эти значения (1250) складываются вместе и полученный результат 2500 затем добавляется к 2850, образуя 5350 как для `WARD`, так и для `MARTIN`. Указывая в операторе `ORDER BY` комбинацию столбцов, которая исключает дубликаты значений (например, любая комбинация значений `SAL` и `EMPNO` является уникальной), мы обеспечиваем правильный ход вычисления текущей суммы.

7.7. Вычисление текущего произведения

ЗАДАЧА

Требуется вычислять текущее произведение для значений числового столбца. Эта задача аналогична задаче из *рецепта 7.6*, но значения не складываются, а перемножаются.

РЕШЕНИЕ

В примере решения вычисляются текущие произведения заработных плат служащих. Практической пользы от этой задачи нет, однако самому описанному здесь методу можно легко найти другие, более полезные применения.

Текущее произведение вычисляем с помощью оконной функции `SUM OVER`, эмулируя умножение сложением логарифмов:

```

1 select empno,ename,sal,
2       exp(sum(ln(sal))over(order by sal,empno)) as running_prod
3 from emp
4 where deptno = 10

```

EMPNO	ENAME	SAL	RUNNING_PROD
7934	MILLER	1300	1300
7782	CLARK	2450	3185000
7839	KING	5000	15925000000

В SQL (или, более формально, в математике) вычисление логарифмов значений меньших или равных нулю не допускается. При наличии в таблице таких значений нужно предпринять меры, чтобы не допустить их передачу функции `LN`. Для удобства в приведенном решении меры предосторожности для недопущения таких значений и значений `NULL` не предпринимаются, но этому моменту следует уделить должное внимание при разработке практических продуктов. Если избежать отрицательных и нулевых значений не представляется возможным, то это решение может вам не подойти. В то же самое время, если имеются только нулевые значения, но не отрицательные, эта проблема решается добавлением к этим значениям 1, т. к., независимо от основания, логарифм единицы всегда равен нулю.

В SQL Server вместо функции `LN` используется функция `LOG`.

Обсуждение

Решение основано на том, что два числа можно умножить следующим способом:

1. Вычислить натуральный логарифм каждого числа.
2. Сложить эти логарифмы.
3. Возвести полученную сумму в степень e с помощью функции `EXP`.

Здесь следует учитывать, что, как упоминалось ранее, этот метод непригоден для нулевых и отрицательных значений, поскольку они находятся вне диапазона действительных значений для логарифмов в SQL.

Работа функции `SUM OVER` подробно рассматривается в *рецепте 7.6*.

7.8. Сглаживание последовательности значений

ЗАДАЧА

Имеется последовательность периодических значений — например, суммы ежемесячных продаж. Как обычно, данные могут существенно отличаться от одного месяца к другому, и нам нужно определить общую тенденцию. Для этого надо реализовать простое средство сглаживания данных — например, взвешенное скользящее среднее.

Пусть у нас есть следующие данные дневного оборота в долларах:

DATE1	SALES
2020-01-01	647
2020-01-02	561
2020-01-03	741
2020-01-04	978
2020-01-05	1062
2020-01-06	1072
...	...

Здесь вроде бы прослеживается тенденция повышения продаж, но мы знаем, что данные продаж подвержены колебаниям, которые усложняют выявление скрытых тенденций. Возможно, в некоторые дни недели или месяца продажи особенно высокие или низкие. Или же возможно, что вследствие особенностей сбора данных иногда данные о продажах за один день объединяются с данными за следующий день, создавая провал, за которым следует пик. При этом отнести такие данные на правильный день по какой-либо причине не представляется возможным. Поэтому нужно выравнять данные за определенный период, чтобы получить более ясную картину происходящего.

Для этого можно применить метод скользящего среднего, складывая текущее n -е значение с предыдущими $n - 1$ значениями и разделив полученную сумму на n . Отображая для сравнения также и предыдущие значения, можно ожидать получения результата наподобие следующего:

DATE1	sales	salesLagOne	SalesLagTwo	MovingAverage
-----	-----	-----	-----	-----
2020-01-01	647	NULL	NULL	NULL
2020-01-02	561	647	NULL	NULL
2020-01-03	741	561	647	649.667
2020-01-04	978	741	561	760
2020-01-05	1062	978	741	927
2020-01-06	1072	1062	978	1037.333
2020-01-07	805	1072	1062	979.667
2020-01-08	662	805	1072	846.333
2020-01-09	1083	662	805	850
2020-01-10	970	1083	662	905

РЕШЕНИЕ

Среднее значение вычисляется по хорошо известной формуле. Применив с этой формулой простое взвешивание, мы можем сделать ее более подходящей для этой задачи, придав больший вес более свежим значениям. Для вычисления скользящего среднего используем функцию LAG:

```
select date1, sales, lag(sales,1) over(order by date1) as salesLagOne,
lag(sales,2) over(order by date1) as salesLagTwo,
```

```
(sales
+ (lag(sales,1) over(order by datel))
+ lag(sales,2) over(order by datel))/3 as MovingAverage
from sales
```

Обсуждение

Взвешенное скользящее среднее представляет собой один из наиболее простых способов анализа данных временного ряда (т. е. данных, создаваемых через определенные временные интервалы). Это всего лишь один из способов вычисления простого скользящего среднего — вы также можете использовать для этого способ сегментации среднего сдвига. Хотя мы здесь выбрали простую формулу скользящего среднего трех чисел, существуют разные формулы с использованием \equiv в зависимости от обрабатываемых данных — разного количества чисел, что делает этот подход особо эффективным.

Например, простое трехточечное взвешенное скользящее среднее, которое подчеркивает самые последние данные, может быть реализовано с помощью следующего варианта решения с обновленными коэффициентами и знаменателем:

```
select datel, sales, lag(sales,1) over(order by datel),
lag(sales,2) over(order by datel),
((3*sales
+ (2*(lag(sales,1) over(order by datel)))
+ (lag(sales,2) over(order by datel)))/6 as SalesMA
from sales
```

7.9. Вычисление моды

ЗАДАЧА

Требуется вычислить моду (в математике *мода* — это элемент с наибольшим количеством вхождений в рассматриваемый набор данных) значений столбца. Например, найти моду зарплат для отдела 20, которая для показанной далее группы зарплат будет равна 3000:

```
select sal
  from emp
 where deptno = 20
 order by sal
```

```
      SAL
-----
      800
     1100
     2975
     3000
     3000
```

РЕШЕНИЕ

DB2, MySQL, PostgreSQL и SQL Server

Чтобы упростить получение моды, ранжируем количество вхождений значений зарплат посредством оконной функции `DENSE_RANK`:

```

1 select sal
2   from (
3 select sal,
4        dense_rank()over( order by cnt desc) as rnk
5   from (
6 select sal, count(*) as cnt
7   from emp
8  where deptno = 20
9  group by sal
10         ) x
11        ) y
12       )
13  where rnk = 1

```

Oracle

Моду для столбца `SAL` можно получить, используя расширение `KEEP` для агрегатной функции `MAX`. Важно отметить, что в случае наличия нескольких мод (т. е. нескольких значений с одинаковым наибольшим количеством вхождений) этот подход возвращает только одну моду с наибольшим значением зарплаты. Чтобы вернуть все моды, это решение необходимо модифицировать должным образом или же просто использовать решение для DB2. В нашем случае, поскольку мода зарплат отдела 20 с величиной 3000 также является и наибольшей зарплатой, следующего решения будет достаточно:

```

1 select max(sal)
2        keep(dense_rank first order by cnt desc) sal
3   from (
4 select sal, count(*) cnt
5   from emp
6  where deptno=20
7  group by sal
8         )

```

Обсуждение

DB2 и SQL Server

Вложенный запрос `X` возвращает каждое значение столбца `SAL` и количество его вхождений. Полученные результаты ранжируются во вложенном запросе `Y` с помощью функции `DENSE_RANK` (допускающей несколько одинаковых мод).

Ранжирование осуществляется на основе количества вхождений каждого значения `SAL`, как показано в следующем запросе:

```

1 select sal,
2     dense_rank()over(order by cnt desc) as rnk
3   from (
4 select sal,count(*) as cnt
5   from emp
6  where deptno = 20
7   group by sal
8     ) x

```

SAL	RNK
3000	1
800	2
1100	2
2975	2

Внешняя часть запроса просто возвращает строку (строки), для которой значение RNK равно 1.

Oracle

Вложенный запрос возвращает каждое значение столбца SAL и количество его вхождений:

```

select sal, count(*) cnt
  from emp
 where deptno=20
  group by sal

```

SAL	CNT
800	1
1100	1
2975	1
3000	2

На следующем шаге мы определяем моду, используя для этого агрегатную функцию MAX с расширением KEEP. Оператор KEEP содержит три подоператора, DENSE_RANK, FIRST и ORDER BY CNT DESC:

```
keep(dense_rank first order by cnt desc)
```

Эти операторы значительно упрощают определение моды. Оператор KEEP определяет значение SAL для возвращения его функцией MAX по значению CNT, возвращаемому вложенным представлением. Выражение выполняется справа налево. Сначала значения CNT упорядочиваются в нисходящем порядке функцией DENSE_RANK, после чего из этих значений выбирается первое. Как можно видеть, в результирующем множестве вложенного представления значение зарплаты 3000 имеет наибольшее количество вхождений CNT, равное 2. Функция MAX(SAL) возвращает наибольшее значения SAL с наибольшим количеством вхождений CNT — в нашем случае это 3000.

См. также

В рецепте 11.11 более подробно рассматривается расширение `KEEP` для агрегатных функций Oracle.

7.10. Вычисление медианы

ЗАДАЧА

Требуется вычислить медиану (значение среднего элемента упорядоченного множества) для столбца числовых значений — например, найти медиану зарплат для отдела 20, которая для показанной далее группы зарплат будет равна 2975:

```
select sal
  from emp
 where deptno = 20
 order by sal
```

```
SAL
-----
800
1100
2975
3000
3000
```

РЕШЕНИЕ

За исключением решения для Oracle (в котором используются встроенные функции вычисления медианы), в остальных решениях применяются оконные функции, которые обеспечивают более эффективное решение по сравнению с традиционным подходом с самосоединением.

DB2 и PostgreSQL

Для нахождения медианы используем оконную функцию `PERCENTILE_CONT`:

```
1 select percentile_cont(0.5)
2       within group(order by sal)
3   from emp
4  where deptno=20
```

SQL Server

Для нахождения медианы используем оконную функцию `PERCENTILE_CONT`:

```
1 select percentile_cont(0.5)
2       within group(order by sal)
3       over()
4   from emp
5  where deptno=20
```

Это решение аналогично решению для DB2, но требует применения предиката `OVER`.

MySQL

MySQL не поддерживает функцию `PERCENTILE_CONT`, поэтому для нее требуется обходное решение. Один из вариантов — использовать функцию `CUME_DIST` в комбинации с обобщенным табличным выражением, воссоздавая тем самым, по сути, функцию `PERCENTILE_CONT`:

```
with rank_tab (sal, rank_sal) as
(
  select sal, cume_dist() over (order by sal)
        from emp
        where deptno=20
),

inter as
(
  select sal, rank_sal from rank_tab
  where rank_sal>=0.5
union
  select sal, rank_sal from rank_tab
  where rank_sal<=0.5
)

select avg(sal) as MedianSal
from inter
```

Oracle

Используем функцию `MEDIAN` или `PERCENTILE_CONT`:

```
1 select median(sal)
2   from emp
3  where deptno=20

1 select percentile_cont(0.5)
2       within group(order by sal)
3   from emp
4  where deptno=20
```

Обсуждение

Oracle, PostgreSQL, SQL Server и DB2

За исключением функции `MEDIAN` для Oracle, структура всех этих решений одинакова. Функция `PERCENTILE_CONT` позволяет напрямую применять определения медианы — 50-й перцентиль. Следовательно, чтобы определить медиану, функции просто передается в параметре значение 0,5.

Конечно же, функция может находить и другие перцентили. Например, 5-й и/или 95-й, чтобы определить выбросы (альтернативный способ определения выбросов

вкратце излагается далее — в *рецепте 7.16* этой главы при рассмотрении среднего абсолютного отклонения).

MySQL

СУБД MySQL не поддерживает функцию `PERCENTILE_CONT`, что делает решение немного сложнее. Чтобы определить медиану, значения `SAL` нужно упорядочить по возрастанию. Эту задачу выполняет функция `CUME_DIST`, заодно обозначая перцентиль каждой строки. Соответственно, эта функция может обеспечить такой же результат, как и функция `PERCENTILE_CONT` в других решениях.

Но при этом нужно иметь в виду, что функцию `CUME_DIST` нельзя размещать в выражении `WHERE`, и поэтому ее нужно применять в обобщенном табличном выражении.

Кроме этого, четное количество строк обуславливает отсутствие строки точно по медиане. Поэтому в решении вычисляется среднее двух значений: наибольшего значения ниже или равного медиане и наименьшего значения выше или равного медиане. Этот подход работает как с четным, так и с нечетным количеством строк. В последнем случае вычисляется среднее двух одинаковых чисел.

7.11. Вычисление процентной доли от целого

ЗАДАЧА

Требуется определить процентную долю суммы определенной группы значений столбца от общей суммы столбца — например, вычислить процентную долю суммы зарплат отдела 10 от общей суммы зарплат.

РЕШЕНИЕ

По большому счету, процесс вычисления процентного значения в SQL ничем не отличается от ручного вычисления: сначала просто делим долю на целое, а затем умножаем на 100. Таким образом, чтобы вычислить процентное отношение суммы зарплат служащих отдела 10, сначала вычисляем эту сумму, а затем делим ее на общую сумму зарплат. И в завершение умножаем полученный результат на 100, чтобы получить процентное значение.

MySQL и PostgreSQL

Сначала делим сумму зарплат `DEPTNO 10` на сумму всех зарплат, а затем умножаем на 100:

```
1 select (sum(
2     case when deptno = 10 then sal end)/sum(sal)
3     )*100 as pct
4 from emp
```

DB2, Oracle и SQL Server

Во вложенном запросе вычисляем сумму всех зарплат и сумму зарплат отдела 10, используя оконную функцию `SUM OVER`. Затем во внешнем запросе выполняем операции деления и умножения:

```
1 select distinct (d10/total)*100 as pct
2   from (
3 select deptno,
4        sum(sal)over() total,
5        sum(sal)over(partition by deptno) d10
6   from emp
7        ) x
8  where deptno=10
```

Обсуждение

MySQL и PostgreSQL

Оператор `CASE` возвращает зарплаты только для отдела `DEPTNO 10`. Затем эти зарплаты суммируются и делятся на сумму всех зарплат. Так как агрегатные функции игнорируют значения `NULL`, конструкция `ELSE` в операторе `CASE` не требуется. Просмотреть значения делимого и делителя можно, исполнив запрос без операции деления:

```
select sum(case when deptno = 10 then sal end) as d10,
       sum(sal)
  from emp
```

```
D10  SUM(SAL)
----  -
8750  29025
```

В зависимости от заданного типа данных значений `SAL` при делении может понадобиться выполнить явное приведение типов, чтобы обеспечить правильный тип данных. Например, если в `DB2`, `SQL Sever` и `PostgreSQL` для столбца `SAL` задан целочисленный тип данных, десятичное значение можно вернуть, используя выражение `CAST` следующим образом:

```
select (cast(
        sum(case when deptno = 10 then sal end)
        as decimal)/sum(sal)
       )*100 as pct
  from emp
```

DB2, Oracle и SQL Server

Как альтернатива традиционному решению в этом решении для вычисления процентной доли от целого используются оконные функции. Если в `DB2` и `SQL Server` для столбца `SAL` задан целочисленный тип данных, прежде чем выполнять деление, нужно выполнить явное приведение типов (`CAST`):

```

select distinct
    cast(d10 as decimal)/total*100 as pct
  from (
select deptno,
    sum(sal)over() total,
    sum(sal)over(partition by deptno) d10
  from emp
    ) x
where deptno=10

```

Важно иметь в виду, что оконные функции применяются после обработки выражения WHERE. Следовательно, фильтрацию значений по номеру отдела DEPTNO нельзя выполнять во вложенном запросе X. Рассмотрим результаты исполнения вложенного запроса X с фильтрацией по DEPTNO и без таковой. Сначала без фильтрации:

```

select deptno,
    sum(sal)over() total,
    sum(sal)over(partition by deptno) d10
  from emp

```

DEPTNO	TOTAL	D10
10	29025	8750
10	29025	8750
10	29025	8750
20	29025	10875
20	29025	10875
20	29025	10875
20	29025	10875
20	29025	10875
30	29025	9400
30	29025	9400
30	29025	9400
30	29025	9400
30	29025	9400

А теперь с фильтрацией:

```

select deptno,
    sum(sal)over() total,
    sum(sal)over(partition by deptno) d10
  from emp
where deptno=10

```

DEPTNO	TOTAL	D10
10	8750	8750
10	8750	8750
10	8750	8750

Так как оконные функции вызываются после выражения `WHERE`, значение `TOTAL` представляет сумму зарплат только для отдела 10. Но для решения этой задачи значение `TOTAL` должно представлять сумму всех зарплат. Поэтому фильтрация по номеру отдела `DEPTNO` должна выполняться вне вложенного запроса `X`.

7.12. Агрегация столбцов, содержащих значения `NULL`

ЗАДАЧА

Требуется выполнить агрегацию столбца, который может содержать значения `NULL`. При этом необходимо сохранить точность агрегации, которая может быть нарушена вследствие игнорирования значений `NULL` агрегатными функциями. Например, надо вычислить среднее значение комиссионных для служащих отдела 30, но некоторые из служащих этого отдела не работают на условиях комиссии (т. е. значение `COMM` для них равно `NULL`). Поскольку агрегатные функции игнорируют значение `NULL`, точность полученного результата будет сомнительной. Поэтому нужно каким-то образом учитывать значения `NULL` в процессе агрегации.

РЕШЕНИЕ

Чтобы включить значения `NULL` в агрегацию, преобразовываем их в 0 с помощью функции `COALESCE`:

```
1 select avg(coalesce(comm,0)) as avg_comm
2   from emp
3  where deptno=30
```

Обсуждение

При работе с агрегатными функциями важно не забывать, что они игнорируют значения `NULL`. Рассмотрим результат решения, полученный без применения функции `COALESCE`:

```
select avg(comm)
   from emp
  where deptno=30
```

```
AVG (COMM)
-----
          550
```

Результат запроса показывает среднее значение комиссионных для отдела 30 равное 550, но по результатам этого запроса:

```
select ename, comm
   from emp
  where deptno=30
 order by comm desc
```

ENAME	COMM
-----	-----
BLAKE	
JAMES	
MARTIN	1400
WARD	500
ALLEN	300
TURNER	0

можно видеть, что только четыре из шести сотрудников работают на условиях комиссии. Сумма всех комиссионных для отдела 30 составляет 2200, и среднее должно быть $2200/6 = 366,66$, а не $2200/4 = 550$. Запрос без функции `COALESCE` дает нам среднее значение комиссионных только для тех служащих отдела 30, которые работают на условиях комиссии, а не для всех служащих этого отдела. Поэтому при работе с агрегатными функциями не забывайте обрабатывать значения `NULL` должным образом.

7.13. Вычисление среднего без учета крайних значений

ЗАДАЧА

Требуется вычислить среднее значение, не учитывая наибольшее и наименьшее значения, чтобы попытаться уменьшить асимметрию распределения. На языке статистики такое среднее называется *усеченным*. Попробуем вычислить среднюю зарплату всех служащих, не учитывая наибольшее и наименьшее ее значения.

РЕШЕНИЕ

MySQL и PostgreSQL

Исключаем крайние значения из вычислений, используя подзапросы:

```

1 select avg(sal)
2   from emp
3  where sal not in (
4     (select min(sal) from emp),
5     (select max(sal) from emp)
6  )

```

DB2, Oracle и SQL Server

Используем функции `MAX OVER` и `MIN OVER` во вложенном подзапросе для создания результирующего множества, из которого можно без труда исключить минимальное и максимальное значения:

```

1 select avg(sal)
2   from (
3 select sal, min(sal)over() min_sal, max(sal)over() max_sal

```

```

4  from emp
5      ) x
6  where sal not in (min_sal,max_sal)

```

Обсуждение

MySQL и PostgreSQL

Подзапросы возвращают наибольшее и наименьшее значения столбца зарплат. Применяем к возвращенным значениям `NOT IN`, исключая их тем самым из вычисления среднего. Не забывайте, что в случае наличия дубликатов предельных значений (т. е. когда несколько служащих имеют одинаковую наибольшую или наименьшую зарплату) они будут не учтены при вычислении среднего. Если нужно исключить только одно вхождение предельных значений, просто вычитаем их из суммы, а затем выполняем деление:

```

select (sum(sal)-min(sal)-max(sal))/(count(*)-2)
      from emp

```

DB2, Oracle и SQL Server

Вложенный запрос `X` возвращает все значения зарплат вместе с наибольшим и наименьшим значениями:

```

select sal, min(sal)over() min_sal, max(sal)over() max_sal
      from emp

```

SAL	MIN_SAL	MAX_SAL
800	800	5000
1600	800	5000
1250	800	5000
2975	800	5000
1250	800	5000
2850	800	5000
2450	800	5000
3000	800	5000
5000	800	5000
1500	800	5000
1100	800	5000
950	800	5000
3000	800	5000
1300	800	5000

Определить максимальное и минимальное значение зарплаты не представляет никаких трудностей, поскольку каждая строка содержит эти значения. Внешний запрос фильтрует строки, возвращенные вложенным запросом `X`, исключая из вычисления среднего строки, в которых значение зарплаты совпадает или с `MIN_SAL`, или с `MAX_SAL`.



Робастная статистика

На языке статистики среднее, вычисленное без учета максимального и минимального значений, называется усеченным средним. Такое среднее считается более надежным и представляет пример *робастного* (устойчивого) статистического вычисления, вследствие меньшей чувствительности к таким проблемам, как статистическое смещение. В *рецепте 7.16* приводится еще один пример робастного статистического средства. Оба эти примера будут полезны пользователям при анализе данных реляционных СУБД, поскольку для них не требуется делать аналитических предположений, проверить которые посредством сравнительно ограниченного набора статистических средств в SQL достаточно сложно.

7.14. Преобразование буквенно-цифровых строк в числа

ЗАДАЧА

Требуется обработать буквенно-цифровые данные, возвратив только цифровую составляющую. Например, строка `paul123f321` должна вернуть `123321`.

РЕШЕНИЕ

DB2

Для извлечения числовых символов из буквенно-цифровой строки используем встроенные функции `TRANSLATE` и `REPLACE`:

```
1 select cast(
2     replace(
3     translate( 'paul123f321',
4         repeat('#',26),
5         'abcdefghijklmnopqrstuvwxyz'),'#','')
6     as integer ) as num
7 from t1
```

Oracle, SQL Server и PostgreSQL

Для извлечения числовых символов из буквенно-цифровой строки используем встроенные функции `TRANSLATE` и `REPLACE`:

```
1 select cast(
2     replace(
3     translate( 'paul123f321',
4         'abcdefghijklmnopqrstuvwxyz',
5         rpad('#',26,'#')),'#','')
6     as integer ) as num
7 from t1
```

MySQL

На момент подготовки материала этой книги MySQL не поддерживает функцию `TRANSLATE`, поэтому решения для этой СУБД не предоставляется.

Обсуждение

Оба приведенных решения отличаются друг от друга только синтаксисом. В частности, в DB2 применяется функция `REPEAT` вместо `RPAD` в Oracle, SQL Server и PostgreSQL, и параметры в списке `TRANSLATE` указаны в DB2 в другом порядке. Следующее объяснение ориентировано на решение для Oracle и PostgreSQL, но также применимо и к решению для DB2. Исполняя запрос по частям, начиная только с вложенной функции `TRANSLATE`, можно видеть, что все очень просто. Сначала функция `TRANSLATE` заменяет все нечисловые символы символом `#`:

```
select translate( 'paul123f321',
'abcdefghijklmnopqrstuvwxyz',
rpad('#',26,'#')) as num
from t1
```

```
          NUM
-----
#####123#321
```

Теперь, когда все нечисловые символы представлены символом `#`, просто удаляем их посредством функции `REPLACE`, а затем выполняем приведение типов `CAST`, чтобы вернуть числовой результат.

Приведенный в задании пример крайне прост, т. к. исходные данные содержат только буквенно-числовые символы. Если исходная строка, кроме буквенно-цифровых, содержит и другие символы, то вместо выискивания их по отдельности можно применить другой подход. А именно: вместо поиска и удаления нечисловых символов выполняем поиск всех цифровых символов и удаляем все другие символы, не являющиеся ими. Следующий пример поможет прояснить этот метод:

```
select replace(
    translate('paul123f321',
    replace(translate( 'paul123f321',
    '0123456789',
    rpad('#',10,'#')), '#', ''),
    rpad('#',length('paul123f321'),'#'),'#', '') as num
from t1
```

```
          NUM
-----
    123321
```

Это решение выглядит немного более запутанным, чем первое, но, разбив его на части, мы увидим, что ничего особо сложного в нем нет. Посмотрим на результаты первого вложенного вызова функции `TRANSLATE`:

```
select translate( 'paul123f321',
    '0123456789',
    rpad('#',10,'#'))
from t1
```

```

TRANSLATE ('
-----
paul###f###

```

Как видим, здесь первым шагом вместо замены нечисловых символов символом # мы заменяем этим символом все числовые символы. На следующем шаге удаляем все вхождения символа #, оставляя только нечисловые символы:

```

select replace(translate( 'paul123f321',
                        '0123456789',
                        rpad('#',10,'#')), '#', '')
from t1

```

```

REPLA
-----
paulf

```

Затем снова вызываем функцию `TRANSLATE` — на этот раз чтобы заменить в исходной строке все нечисловые символы (возвращенные предыдущим запросом) символом #:

```

select translate('paul123f321',
                replace(translate( 'paul123f321',
                                '0123456789',
                                rpad('#',10,'#')), '#', ''),
                rpad('#',length('paul123f321'),'#'))
from t1

```

```

TRANSLATE ('
-----
####123#321

```

На этом этапе остановимся и рассмотрим внешний вызов функции `TRANSLATE`. Во втором параметре функции `RPAD` (или функции `REPEAT` в решении для DB2) передается длина исходной строки. Это удобная величина, поскольку ни один символ не может входить в строку большее количество раз, чем длина строки, частью которой он является. Заменяв все нечисловые символы символом #, на последнем шаге удаляем эти символы с помощью функции `REPLACE`. В результате получаем только цифровые символы.

7.15. Изменение значений в текущей сумме

ЗАДАЧА

Требуется модифицировать значения в текущей сумме в зависимости от значений в каком-либо другом столбце. Например, надо отобразить историю транзакций кредитной карточки вместе с текущим балансом после каждой транзакции. Для демонстрации примера мы воспользуемся следующим представлением:

```

create view V (id,amt,trx)
as
select 1, 100, 'PR' from t1 union all

```

```

select 2, 100, 'PR' from t1 union all
select 3, 50, 'PY' from t1 union all
select 4, 100, 'PR' from t1 union all
select 5, 200, 'PY' from t1 union all
select 6, 50, 'PY' from t1

```

```
select * from V
```

```

ID      AMT TRX
--  -----
1       100 PR
2       100 PR
3        50 PY
4       100 PR
5       200 PY
6        50 PY

```

Каждая транзакция однозначно идентифицируется значением в столбце ID. Столбец AMT содержит сумму каждой транзакции (покупку или платеж). Значения столбца TRX определяют тип транзакции: PY означает платеж (payment), а PR — покупку (purchase). Для значений PY текущее значение AMT нужно вычесть из текущей суммы, а для значений PR — добавить к ней. В конечном итоге результирующее множество должно иметь следующий вид:

```

TRX_TYPE      AMT      BALANCE
-----  -----
PURCHASE      100       100
PURCHASE      100       200
PAYMENT        50       150
PURCHASE      100       250
PAYMENT       200        50
PAYMENT        50         0

```

РЕШЕНИЕ

Используем оконную функцию SUM OVER для создания текущей суммы и выражение CASE для определения типа транзакции:

```

1 select case when trx = 'PY'
2           then 'PAYMENT'
3           else 'PURCHASE'
4         end trx_type,
5         amt,
6         sum(
7           case when trx = 'PY'
8             then -amt else amt
9           end
10        )over (order by id,amt) as balance
11 from V

```

Обсуждение

Выражение CASE определяет, что делать с текущим значением AMT: добавлять ли его или вычитать из текущей суммы. Для платежных транзакций значение AMT меняется на отрицательное, тем самым уменьшая текущую сумму. Далее приводится результат применения выражения CASE:

```
select case when trx = 'PY'
         then 'PAYMENT'
         else 'PURCHASE'
       end trx_type,
       case when trx = 'PY'
         then -amt else amt
       end as amt
from V
```

TRX_TYPE	AMT
PURCHASE	100
PURCHASE	100
PAYMENT	-50
PURCHASE	100
PAYMENT	-200
PAYMENT	-50

После определения типа транзакции значения AMT добавляются или вычитаются из текущей суммы. Создание текущей суммы с помощью оконной функции SUM OVER или скалярного подзапроса подробно рассматривается в *рецепте 7.6*.

7.16. Находим выбросы, используя среднее абсолютное отклонение

ЗАДАЧА

Требуется выявить потенциально недействительные данные. Данные могут быть недействительными по многим причинам — например, при ошибке во время сбора данных, вызванной, допустим, сбоем записывающего данные датчика. Ошибка также может быть допущена при ручном вводе данных. Возможно и возникновение при генерировании данных нестандартных ситуаций, позволяющих предполагать, что данные правильные, но все равно надо быть осторожным в принятии какого-либо заключения на их основе. Таким образом, нам нужно выявить выбросы среди собранных данных.

Во многих курсах, по статистике, неспециалистов учат распространенному способу выявления выбросов, заключающемуся в вычислении стандартного отклонения данных и объявлении выбросами всех точек данных, превышающих три стандартных отклонения (или другое подобное значение). Но в случае отсутствия нормаль-

ного распределения данных этот метод может ошибочно определять выбросы, особенно при несимметричном распределении данных, или если разрежение данных не следует разрежению нормального распределения при удалении от среднего.

РЕШЕНИЕ

Сначала вычисляем медиану значений, используя соответствующий рецепт, рассмотренный ранее в этой главе (см. *рецепт 7.10*). Этот запрос нужно делать в обобщенном табличном выражении, чтобы его результаты были доступны для последующих запросов. Отклонение — это абсолютная разница между медианой и каждым значением, а медианное абсолютное отклонение — это медиана этих отклонений, поэтому нам опять нужно вычислить медиану.

SQL Server

СУБД SQL Server поддерживает функцию `PERCENTILE_CONT`, что упрощает нахождение медианы. Поскольку нам надо вычислить и работать с двумя разными медианами, нам потребуется последовательность обобщенных табличных выражений:

```
with median (median)
as
(select distinct percentile_cont(0.5) within group(order by sal)
    over()
from emp),

Deviation (Deviation)
as
(Select abs(sal-median)
from emp join median on 1=1),

MAD (MAD) as
(select DISTINCT PERCENTILE_CONT(0.5) within group(order by deviation)
    over()
from Deviation )

select abs(sal-MAD)/MAD, sal, ename, job
from MAD join emp on 1=1
```

PostgreSQL и DB2

Здесь общий шаблон решения такой же, но для функции `PERCENTILE_CONT` нужен другой синтаксис, поскольку PostgreSQL и DB2 работают с этой функцией как с агрегатной, а не оконной:

```
with median (median)
as
(select percentile_cont(0.5) within group(order by sal)
from emp),
```

```

devtab (deviation)
  as
(select abs(sal-median)
from emp join median),

MedAbsDeviation (MAD) as
(select percentile_cont (0.5) within group(order by deviation)
from devtab)

select abs(sal-MAD)/MAD, sal, ename, job
FROM MedAbsDeviation join emp

```

Oracle

Наличие в Oracle поддержки функции `MEDIAN` упрощает решения для этой СУБД. Но для обработки скалярного значения отклонения все равно нужно использовать обобщенное табличное выражение:

```

with
Deviation (Deviation)
  as
(select abs(sal-median(sal))
from emp),

MAD (MAD) as
(select median(Deviation)
from Deviation )

select abs(sal-MAD)/MAD, sal, ename, job
FROM MAD join emp

```

MySQL

Как уже упоминалось ранее, MySQL не поддерживает функции `MEDIAN` и `PERCENTILE_CONT`. Поэтому для определения обеих медиан, требуемых для вычисления медианного абсолютного отклонения, нужно выполнить два запроса, размещенных в обобщенном табличном выражении. В результате решение получается несколько растянутым:

```

with rank_tab (sal, rank_sal) as (
select sal, cume_dist() over (order by sal)
from emp),
inter as
(
select sal, rank_sal from rank_tab
where rank_sal>=0.5
union
select sal, rank_sal from rank_tab
where rank_sal<=0.5
)
,

```

```
medianSal (medianSal) as

(
select (max(sal)+min(sal))/2
from inter),
deviationSal (Sal,deviationSal) as
(select Sal,abs(sal-medianSal)
from emp join medianSal
on l=1
)
,

distDevSal (sal,deviationSal,distDeviationSal) as

(
select sal,deviationSal,cume_dist() over (order by deviationSal)
from deviationSal
),

DevInter (DevInter, sal) as
(
select min(deviationSal), sal
from distDevSal
where distDeviationSal >= 0.5

union

select max(DeviationSal), sal
from distDevSal
where distDeviationSal <= 0.5
),

MAD (MedianAbsoluteDeviance) as
(
select abs(emp.sal-(min(devInter)+max(devInter))/2)
from emp join DevInter on l=1
)

select emp.sal,MedianAbsoluteDeviance,
(emp.sal-deviationSal)/MedianAbsoluteDeviance
from (emp join MAD on l=1)
join deviationSal on emp.sal=deviationSal.sal
```

Обсуждение

Все приведенные здесь решения основаны на сходной стратегии. Сначала вычисляется медиана, а затем медианное абсолютное отклонение, т. е. медиана разниц между каждым значением и медианой. Затем выполняется запрос для вычисления

отношения отклонения каждого значения к медианному отклонению. На этом этапе полученный результат можно использовать в качестве стандартного отклонения. Например, если значение отстоит от медианы на три или больше отклонений, его можно считать выбросом.

Как упоминалось ранее, преимущество описанного подхода над использованием стандартного отклонения состоит в том, что полученная таким образом интерпретация действительна даже при ненормальном распределении данных. Например, в случае перекошенного в одну сторону распределения медианное абсолютное отклонение все равно предоставит действительный ответ.

В используемом нами наборе зарплат только одно значение зарплаты больше, чем три абсолютных отклонения от медианы, — зарплата президента компании.

Хотя разница между размером зарплат президентов и зарплатами большинства остальных работников может вызывать неоднозначную реакцию, то, что аномальная зарплата принадлежит президенту компании, соответствует нашему пониманию данных. При других обстоятельствах отсутствие удовлетворительного объяснения такого большого отличия этого значения от других могло бы вызвать вопросы относительно правильности этого значения или его соответствия другим значениям. Например, если значение в действительности правильное, оно может заставить нас думать о необходимости анализировать данные по нескольким подгруппам.



Многие обычные статистические средства — например, среднее и стандартное отклонение — предполагают нормальное распределение данных по кривой колокола. Вместе с тем многие наборы данных действительно соответствуют такому распределению, но многие другие — нет.

Существует несколько способов проверки — как визуальных, так и вычислительных, — соответствует ли набор данных нормальному распределению. Программы обработки статистических данных обычно поддерживают функции для таких проверок, но в SQL подобные функции отсутствуют и их трудно воспроизвести другими доступными средствами. Однако существуют альтернативные статистические инструменты, которые не предполагают определенного распределения данных. Это непараметрические статистические инструменты, и они более безопасны.

7.17. Обнаруживаем аномальные значения, используя закон Бенфорда

ЗАДАЧА

Хотя такие аномальные данные, как выбросы, легко поддаются обнаружению, что и было продемонстрировано в предыдущем рецепте, идентификация других типов аномальностей может оказаться более трудной задачей. Один из способов определить наличие аномальных данных при отсутствии очевидных признаков выбросов состоит в вычислении частоты вхождений цифр, которая обычно должна следовать закону Бенфорда. Хотя закон Бенфорда наиболее часто применяется для обнаружения мошенничества в ситуациях, когда люди добавляют в набор данных ложные цифры, его также можно использовать для решения более общей задачи обнаруже-

ния данных, не следующих ожидаемым закономерностям. Например, с помощью этого закона можно обнаруживать такие ошибки, как дубликаты данных, которые не обязательно будут выделяться как выбросы.

РЕШЕНИЕ

Применение закона Бенфорда состоит в сравнении вычисленного ожидаемого распределения цифр с действительным. Хотя в более утонченных анализах исследуются первая и вторая цифры и комбинации цифр, в этом примере мы будем работать только с первыми цифрами каждого значения.

Для решения мы сравним частоту данных, предсказанную законом Бенфорда, с их действительной частотой. Результат решения должен выводиться в четырех столбцах: первая цифра значения, количество вхождений первой цифры, частота первых цифр по закону Бенфорда и их настоящая частота:

```
with
FirstDigits (FirstDigit)
as
(select left(cast(SAL as CHAR),1) as FirstDigit
 from emp),

TotalCount (Total)
as
(select count(*)
 from emp),

ExpectedBenford (Digit,Expected)
as
(select value, (log10(value + 1) - log10(value)) as expected
 from t10
 where value < 10)

select count(FirstDigit),Digit
,coalesce(count(*)/Total,0) as ActualProportion,Expected
From FirstDigits
Join TotalCount
Right Join ExpectedBenford
on FirstDigits.FirstDigit=ExpectedBenford.Digit
group by Digit
order by Digit;
```

Обсуждение

В решении нам нужно подсчитывать две разные величины: общее количество строк и количество строк, содержащих разные первые цифры. Это обуславливает необходимость использования обобщенного табличного выражения. Строго говоря, за-

прос для получения ожидаемых результатов по закону Бенфорда не обязательно помещать отдельно в обобщенном табличном выражении. Но в нашем решении этот подход применяется потому, что это позволяет идентифицировать цифры с нулевым количеством вхождений и отображать их в таблице, используя правое объединение.

Подсчет первых цифр также можно выполнить в главном запросе, но мы решили не делать этого, чтобы повысить удобство чтения, не включив выражение `LEFT(CAST(...` в предикат `GROUP BY`.

Закон Бенфорда основан на простых математических вычислениях:

Ожидаемая частота = $\log_{10}((d+1)/1)$

Сначала генерируем соответствующие значения, используя для этого сводную таблицу T10. Затем нужно просто вычислить настоящую частоту первых цифр, чтобы сравнить ее с вычисленной ожидаемой частотой. Для этого надо предварительно идентифицировать первые цифры.

Закон Бенфорда лучше всего работает со сравнительно большими наборами данных, а также когда диапазон значений охватывает несколько порядков величин (10, 100, 1000 и т. д.). В нашем случае эти условия были не совсем выдержаны. В то же самое время отклонение от ожидаемого значения все равно должно вызвать у нас подозрение, что исследуемые данные являются в некотором смысле фиктивными и заслуживают дальнейшей проверки.

7.18. Подведем итоги

Поскольку данные предприятий часто хранятся в базах данных с поддержкой SQL, использование SQL для интерпретации этих данных вполне оправданно. Язык SQL не поддерживает полный набор статистических инструментов, которыми обладают специальные средства, такие как аналитическое программное обеспечение SAS, язык программирования для статистических вычислений R или статистические библиотеки языка Python. Тем не менее он обладает большим набором инструментов для вычислений, которые, как мы видели, могут обеспечить более глубокое понимание статистических свойств используемых данных.

Арифметические операции с датами

В этой главе мы рассмотрим способы выполнения простых арифметических операций с датами. Рецепты охватывают круг общих задач — таких, как добавление дней к датам, вычисление количества рабочих дней между датами и вычисление разницы в днях между датами.

Возможность успешной работы с датами с помощью встроенных функций используемой СУБД может значительно повысить вашу производительность. Во всех рецептах этой главы мы пытаемся использовать встроенные функции каждой из рассматриваемых СУБД. Кроме этого, для всех рецептов мы решили использовать один формат даты: DD-MON-YYYY (ДД-МЕС-ГГГГ). Конечно же, существует несколько других распространенных форматов даты — например, стандартный формат ISO: DD-MM-YYYY (ДД-ММ-ГГГГ).

Однако формат ДД-МЕС-ГГГГ был принят нами в качестве стандарта, чтобы предоставить читателям, работающим только с одной СУБД, возможность познакомиться с некоторыми особенностями других баз данных. Работая с одним стандартным форматом, вы сможете сосредоточиться на разных методах и функциях каждой СУБД, не беспокоясь об их форматах данных по умолчанию.



В этой главе основное внимание уделяется базовым арифметическим операциям с датами. Более продвинутые рецепты для работы с датами рассматриваются в следующей главе. В рецептах этой главы используются простые типы дат. Если вы работаете с более сложными типами дат, вам нужно будет откорректировать приведенные здесь решения соответствующим образом.

8.1. Сложение и вычитание дней, месяцев и лет

ЗАДАЧА

Требуется добавить или вычесть определенное количество дней, месяцев или лет от какой-либо даты. Например, для даты приема на работу `HIREDATE` служащего `CLARK` надо получить шесть разных дат: пять дней, пять месяцев и пять лет до и после этой даты. Служащий `CLARK` был принят на работу `09-JUN-1981`, поэтому результирующее множество должно иметь следующий вид:

```

HD_MINUS_5D HD_PLUS_5D  HD_MINUS_5M HD_PLUS_5M  HD_MINUS_5Y HD_PLUS_5Y
-----
04-JUN-1981 14-JUN-1981 09-JAN-1981 09-NOV-1981 09-JUN-1976 09-JUN-1986

```

РЕШЕНИЕ

DB2

Стандартные операции сложения и вычитания можно также использовать и с датами, но после любого значения, добавляемого или вычитаемого из даты, нужно указывать единицу времени, которое оно представляет:

```

1 select hiredate -5 day as hd_minus_5D,
2     hiredate +5 day as hd_plus_5D,
3     hiredate -5 month as hd_minus_5M,
4     hiredate +5 month as hd_plus_5M,
5     hiredate -5 year as hd_minus_5Y,
6     hiredate +5 year as hd_plus_5Y
7   from emp
8  where deptno = 10

```

Oracle

Для дней используем стандартные операторы сложения и вычитания, а для месяцев и лет — функцию `ADD_MONTHS`:

```

1 select hiredate-5 as hd_minus_5D,
2     hiredate+5 as hd_plus_5D,
3     add_months(hiredate,-5) as hd_minus_5M,
4     add_months(hiredate,5) as hd_plus_5M,
5     add_months(hiredate,-5*12) as hd_minus_5Y,
6     add_months(hiredate,5*12) as hd_plus_5Y
7   from emp
8  where deptno = 10

```

PostgreSQL

Используем стандартные операторы сложения и вычитания в сочетании с ключевым словом `INTERVAL`, указывая при этом требуемую единицу времени. Добавляемое или вычитаемое значение совместно с единицей времени нужно заключать в одинарные кавычки:

```

1 select hiredate - interval '5 day' as hd_minus_5D,
2     hiredate + interval '5 day' as hd_plus_5D,
3     hiredate - interval '5 month' as hd_minus_5M,
4     hiredate + interval '5 month' as hd_plus_5M,
5     hiredate - interval '5 year' as hd_minus_5Y,
6     hiredate + interval '5 year' as hd_plus_5Y
7   from emp
8  where deptno=10

```

MySQL

Используем стандартные операторы сложения и вычитания в сочетании с ключевым словом `INTERVAL`, указывая при этом требуемую единицу времени. В отличие от решения для PostgreSQL, значение и единица времени в кавычки не заключаются:

```
1 select hiredate - interval 5 day as hd_minus_5D,
2     hiredate + interval 5 day as hd_plus_5D,
3     hiredate - interval 5 month as hd_minus_5M,
4     hiredate + interval 5 month as hd_plus_5M,
5     hiredate - interval 5 year as hd_minus_5Y,
6     hiredate + interval 5 year as hd_plus_5Y
7   from emp
8  where deptno=10
```

В качестве альтернативы можно использовать также функцию `DATE_ADD`:

```
1 select date_add(hiredate,interval -5 day) as hd_minus_5D,
2     date_add(hiredate,interval 5 day) as hd_plus_5D,
3     date_add(hiredate,interval -5 month) as hd_minus_5M,
4     date_add(hiredate,interval 5 month) as hd_plus_5M,
5     date_add(hiredate,interval -5 year) as hd_minus_5Y,
6     date_add(hiredate,interval 5 year) as hd_plus_5DY
7   from emp
8  where deptno=10
```

SQL Server

Функция `DATEADD` позволят добавлять к дате разные единицы времени и вычитать их из нее:

```
1 select dateadd(day,-5,hiredate) as hd_minus_5D,
2     dateadd(day,5,hiredate) as hd_plus_5D,
3     dateadd(month,-5,hiredate) as hd_minus_5M,
4     dateadd(month,5,hiredate) as hd_plus_5M,
5     dateadd(year,-5,hiredate) as hd_minus_5Y,
6     dateadd(year,5,hiredate) as hd_plus_5Y
7   from emp
8  where deptno = 10
```

Обсуждение

В решении для Oracle используется то обстоятельство, что при выполнении арифметических операций с датами целочисленные значения представляют дни. Но это касается только операций со значениями типа `DATE`. СУБД Oracle также поддерживает тип дат `TIMESTAMP`. Для операций с датами этого типа нужно использовать решение для PostgreSQL с применением ключевого слова `INTERVAL`. Остерегайтесь также передавать даты типа `TIMESTAMP` устаревшим функциям работы с датами — таким как `ADD_MONTHS`, поскольку это может вызвать потерю информации о дробных долях секунды, которая может в них содержаться.

Ключевое слово `INTERVAL` и сопутствующие ему строковые константы соответствуют SQL-синтаксису стандарта ISO. Согласно этому стандарту значения временных интервалов нужно заключать в одинарные кавычки. СУБД PostgreSQL и Oracle9z и более поздние версии придерживаются этого стандарта, а MySQL немного отклоняется от него, не поддерживая кавычки.

8.2. Вычисление количества дней между двумя датами

ЗАДАЧА

Требуется вычислить разницу в днях между двумя датами. Например, определить разницу между датами приема на работу `HIREDATE` служащих `ALLEN` и `WARD`.

РЕШЕНИЕ

DB2

Используя вложенные запросы, определяем значения `HIREDATE` для `WARD` и `ALLEN`, а затем с помощью функции `DAYS` вычитаем одно значение из другого:

```

1 select days(ward_hd) - days(allen_hd)
2     from (
3 select hiredate as ward_hd
4     from emp
5     where ename = 'WARD'
6         ) x,
7     (
8 select hiredate as allen_hd
9     from emp
10    where ename = 'ALLEN'
11        ) y

```

Oracle и PostgreSQL

Используя вложенные запросы, определяем значения `HIREDATE` для `WARD` и `ALLEN`, а затем вычитаем одно значение из другого:

```

1 select ward_hd - allen_hd
2     from (
3 select hiredate as ward_hd
4     from emp
5     where ename = 'WARD'
6         ) x,
7     (
8 select hiredate as allen_hd
9     from emp
10    where ename = 'ALLEN'
11        ) y

```

MySQL и SQL Server

Для определения количества дней между двумя датами используем функцию DATEDIFF. Версии функции для MySQL нужно передать только два параметра: даты, между которыми нужно вычислить разницу в днях. Во избежание отрицательного результата меньшая из двух дат должна быть первой в списке параметров. Для SQL Server применяется обратный порядок передаваемых дат. Версия функции для SQL Server позволяет указывать требуемый формат возвращаемого значения. В приведенном примере мы хотим получить разницу в днях. Далее приводится версия решения для SQL Server:

```

1 select datediff(day,allen_hd,ward_hd)
2   from (
3 select hiredate as ward_hd
4   from emp
5  where ename = 'WARD'
6       ) x,
7   (
8 select hiredate as allen_hd
9   from emp
10  where ename = 'ALLEN'
11       ) y

```

Для MySQL нужно просто удалить первый аргумент и поменять местами аргументы ALLEN_HD и WARD_HD.

Обсуждение

Во всех решениях даты HIREDATE для служащих WARD и ALLEN возвращаются вложенными запросами X и Y, соответственно. Например:

```

select ward_hd, allen_hd
   from (
select hiredate as ward_hd
   from emp
  where ename = 'WARD'
       ) y,
 (
select hiredate as allen_hd
   from emp
  where ename = 'ALLEN'
       ) x

```

```
WARD_HD ALLEN_HD
```

```
-----
22-FEB-2006 20-FEB-2006
```

Как можно видеть, поскольку между X и Y не указывается объединение, то создается декартово произведение. В нашем случае, поскольку кардинальность как X,

так и Y равна 1, отсутствие объединения не причиняет никакого вреда. Таким образом, в конечном итоге результирующее множество будет состоять из одной строки (что вполне очевидно, поскольку $1 \times 1 = 1$).

Чтобы получить разницу в днях, просто отнимаем одно из возвращенных значений от другого с помощью средств, поддерживаемых используемой базой данных.

8.3. Вычисление количества рабочих дней между двумя датами

ЗАДАЧА

Требуется определить количество рабочих дней между двумя датами, включая сами даты. То есть между, например, понедельником 10 января и вторником 11 января будет два рабочих дня, поскольку эти дни являются обычными рабочими днями. Для этого рецепта имеется в виду, что «рабочий день» — это любой день, кроме субботы или воскресенья.

РЕШЕНИЕ

В рассматриваемых здесь примерах вычисляется количество рабочих дней между датами приема на работу `hiredate` служащих `BLAKE` и `JONES`. В процессе решения используется сводная таблица, возвращающая строку для каждого дня между двумя конечными датами (включая сами эти даты). Количество рабочих дней определяется простым подсчетом возвращенных строк, не являющихся субботой или воскресеньем.



Чтобы исключить также и праздники, можно создать таблицу `HOLIDAYS`, а затем добавить в запрос простой предикат `NOT IN`, чтобы исключить из решения дни, указанные в таблице `HOLIDAYS`.

DB2

Используем сводную таблицу `T500`, чтобы создать количество строк, представляющих дни между двумя датами, а затем подсчитываем строки, не являющиеся субботой или воскресеньем. Название дня для каждой даты получаем с помощью функции `DAYNAME`. Например:

```

1 select sum(case when dayname(jones_hd+t500.id day -1 day)
2               in ( 'Saturday','Sunday' )
3               then 0 else 1
4               end) as days
5   from (
6 select max(case when ename = 'BLAKE'
7               then hiredate
8               end) as blake_hd,
```



```

9         max(case when ename = 'JONES'
10             then hiredate
11             end) as jones_hd
12     from emp
13     where ename in ( 'BLAKE','JONES' )
14           ) x,
15           t500
16     where t500.id <= blake_hd-jones_hd+1

```

MySQL

Используем сводную таблицу T500, чтобы создать количество строк, представляющих дни между двумя датами, а затем подсчитываем строки, не являющиеся субботой или воскресеньем. Чтобы добавить дни к каждой дате, используем функцию DATE_ADD. Название дня для каждой даты получаем с помощью функции DAYNAME:.

```

1 select sum(case when date_format(
2             date_add(jones_hd,
3             interval t500.id-1 DAY),'%a')
4             in ( 'Sat','Sun' )
5             then 0 else 1
6             end) as days
7     from (
8     select max(case when ename = 'BLAKE'
9                 then hiredate
10                end) as blake_hd,
11           max(case when ename = 'JONES'
12                 then hiredate
13                end) as jones_hd
14     from emp
15     where ename in ( 'BLAKE','JONES' )
16           ) x,
17           t500
18     where t500.id <= datediff(blake_hd,jones_hd)+1

```

Oracle

Используем сводную таблицу T500, чтобы создать количество строк, представляющих дни между двумя датами, а затем подсчитываем строки, не являющиеся субботой или воскресеньем. Название дня для каждой даты получаем с помощью функции TO_CHAR:

```

1 select sum(case when to_char(jones_hd+t500.id-1,'DY')
2             in ( 'SAT','SUN' )
3             then 0 else 1
4             end) as days

```

```

5   from (
6   select max(case when ename = 'BLAKE'
7             then hiredate
8             end) as blake_hd,
9             max(case when ename = 'JONES'
10            then hiredate
11            end) as jones_hd
12   from emp
13   where ename in ( 'BLAKE','JONES' )
14            ) x,
15            t500
16   where t500.id <= blake_hd-jones_hd+1

```

PostgreSQL

Используем сводную таблицу T500, чтобы создать количество строк, представляющих дни между двумя датами, а затем подсчитываем строки, не являющиеся субботой или воскресеньем. Название дня для каждой даты получаем с помощью функции TO_CHAR

```

1   select sum(case when trim(to_char(jones_hd+t500.id-1,'DAY'))
2             in ( 'SATURDAY','SUNDAY' )
3             then 0 else 1
4             end) as days
5   from (
6   select max(case when ename = 'BLAKE'
7             then hiredate
8             end) as blake_hd,
9             max(case when ename = 'JONES'
10            then hiredate
11            end) as jones_hd
12   from emp
13   where ename in ( 'BLAKE','JONES' )
14            ) x,
15            t500
16   where t500.id <= blake_hd-jones_hd+1

```

SQL Server

Используем сводную таблицу T500, чтобы создать количество строк, представляющих дни между двумя датами, а затем подсчитываем строки, не являющиеся субботой или воскресеньем. Название дня для каждой даты получаем с помощью функции DAYNAME:

```

1   select sum(case when datename(dw,jones_hd+t500.id-1)
2             in ( 'SATURDAY','SUNDAY' )
3             then 0 else 1
4             end) as days

```

```

5     from (
6 select max(case when ename = 'BLAKE'
7             then hiredate
8             end) as blake_hd,
9         max(case when ename = 'JONES'
10            then hiredate
11            end) as jones_hd
12     from emp
13     where ename in ( 'BLAKE','JONES' )
14           ) x,
15           t500
16     where t500.id <= datediff(day,jones_hd-blake_hd)+1

```

Обсуждение

Хотя для определения названий дней недели в каждой СУБД используются разные встроенные функции, для всех их применяется аналогичный общий подход к решению. Решение можно разбить на два шага:

1. Возвращаем количество дней между начальной и конечной датами (включая дни для самих дат).
2. Подсчитываем количество дней (т. е. строк), исключая выходные.

Первый шаг реализуется вложенным запросом X. Обратите внимание на использование в этом запросе агрегатной функции MAX, удаляющей значения NULL. Чтобы разобраться с работой функции MAX, рассмотрим вложенный запрос X и его результаты без применения этой функции:

```

select case when ename = 'BLAKE'
         then hiredate
        end as blake_hd,
       case when ename = 'JONES'
         then hiredate
        end as jones_hd
  from emp
  where ename in ( 'BLAKE','JONES' )

```

```

BLAKE_HD    JONES_HD
-----
           02-APR-1981
01-MAY-1981

```

Как видим, без применения функции MAX возвращаются две строки. А с применением этой функции вместо двух возвращается только одна строка, и значения NULL удаляются:

```

select max(case when ename = 'BLAKE'
               then hiredate
              end) as blake_hd,

```

```

max(case when ename = 'JONES'
      then hiredate
      end) as jones_hd
from emp
where ename in ( 'BLAKE','JONES' )

```

```

BLAKE_HD      JONES_HD
-----
01-MAY-1981  02-APR-1981

```

В рассматриваемом случае количество дней между этими заданными датами (включая сами даты) составляет 30. Теперь, когда обе даты находятся в одной строке, на следующем шаге генерируем по одной строке для каждого из этих 30 дней, используя для этого таблицу T500. Поскольку каждое следующее значение ID в таблице T500 просто на единицу больше предыдущего значения, чтобы сгенерировать последовательные даты, начиная с JONES_HD до BLAKE_HD (включая эти две даты), добавляем каждую строку, возвращенную таблицей T500, к более ранней из двух дат (т. е. к JONES_HD). Далее приводится соответствующий запрос и результаты его исполнения (используется синтаксис решения для Oracle):

```

select x.*, t500.*, jones_hd+t500.id-1
  from (
select max(case when ename = 'BLAKE'
      then hiredate
      end) as blake_hd,
max(case when ename = 'JONES'
      then hiredate
      end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
    ) x,
  t500
 where t500.id <= blake_hd-jones_hd+1

```

BLAKE_HD	JONES_HD	ID	JONES_HD+T500
01-MAY-1981	02-APR-1981	1	02-APR-1981
01-MAY-1981	02-APR-1981	2	03-APR-1981
01-MAY-1981	02-APR-1981	3	04-APR-1981
01-MAY-1981	02-APR-1981	4	05-APR-1981
01-MAY-1981	02-APR-1981	5	06-APR-1981
01-MAY-1981	02-APR-1981	6	07-APR-1981
01-MAY-1981	02-APR-1981	7	08-APR-1981
01-MAY-1981	02-APR-1981	8	09-APR-1981
01-MAY-1981	02-APR-1981	9	10-APR-1981
01-MAY-1981	02-APR-1981	10	11-APR-1981
01-MAY-1981	02-APR-1981	11	12-APR-1981
01-MAY-1981	02-APR-1981	12	13-APR-1981
01-MAY-1981	02-APR-1981	13	14-APR-1981

01-MAY-1981	02-APR-1981	14	15-APR-1981
01-MAY-1981	02-APR-1981	15	16-APR-1981
01-MAY-1981	02-APR-1981	16	17-APR-1981
01-MAY-1981	02-APR-1981	17	18-APR-1981
01-MAY-1981	02-APR-1981	18	19-APR-1981
01-MAY-1981	02-APR-1981	19	20-APR-1981
01-MAY-1981	02-APR-1981	20	21-APR-1981
01-MAY-1981	02-APR-1981	21	22-APR-1981
01-MAY-1981	02-APR-1981	22	23-APR-1981
01-MAY-1981	02-APR-1981	23	24-APR-1981
01-MAY-1981	02-APR-1981	24	25-APR-1981
01-MAY-1981	02-APR-1981	25	26-APR-1981
01-MAY-1981	02-APR-1981	26	27-APR-1981
01-MAY-1981	02-APR-1981	27	28-APR-1981
01-MAY-1981	02-APR-1981	28	29-APR-1981
01-MAY-1981	02-APR-1981	29	30-APR-1981
01-MAY-1981	02-APR-1981	30	01-MAY-1981

Как можно видеть в выражении `WHERE`, чтобы генерировать требуемые 30 строк, к разнице между `BLAKE_HD` и `JONES_HD` нужно добавить 1 (иначе получим только 29 строк). Также можно видеть, что в списке `SELECT` внешнего запроса из значения `T500.ID` вычитается 1, поскольку значения `ID` начинаются с 1, и добавление 1 к `JONES_HD` вызвало бы исключение этой даты из конечного счета.

Создав требуемое количество строк результирующего множества, с помощью выражения `CASE` помечаем рабочие дни меткой 1, а выходные — меткой 0. И на последнем шаге, используя агрегатную функцию `SUM`, подсчитываем количество единиц, чтобы получить конечный ответ.

8.4. Вычисление количества месяцев или лет между двумя датами

ЗАДАЧА

Требуется вычислить разницу в месяцах или годах между двумя датами. Например, определить количество месяцев между датами приема на работу первого и последнего служащего, а также перевести это значение в количество лет.

РЕШЕНИЕ

Так как год всегда содержит 12 месяцев, чтобы получить количество лет между двумя датами, сначала вычисляем количество месяцев между ними, а затем делим полученное значение на 12. Выполнив эти действия, результат нужно округлить в большую или меньшую сторону, имея в виду следующее соображение. Например, если первое значение `HIREDATE` равно `17-DEC-1980`, а последнее — `12-JAN-1983`, то разница по календарным годам будет 3 года (1983 минус 1980), а вот по месяцам —

25 месяцев (или чуть более двух лет). Поэтому результат вычисления надо соответственно откорректировать. Следующие решения возвращают ответ в месяцах (25 месяцев здесь это примерно два года).

DB2 и MySQL

Используем функции `YEAR` и `MONTH` для возвращения значения года в формате `YYYY` и значения месяца в формате `MM` для указанных дат:

```

1 select mnth, mnth/12
2   from (
3 select (year(max_hd) - year(min_hd))*12 +
4        (month(max_hd) - month(min_hd)) as mnth
5   from (
6 select min(hiredate) as min_hd, max(hiredate) as max_hd
7   from emp
8        ) x
9        ) y

```

Oracle

Для вычисления разницы между двумя датами в месяцах используем функцию `MONTHS_BETWEEN` (чтобы получить разницу в годах, просто делим полученный результат на 12):

```

1 select months_between(max_hd,min_hd),
2        months_between(max_hd,min_hd)/12
3   from (
4 select min(hiredate) min_hd, max(hiredate) max_hd
5   from emp
6        ) x

```

PostgreSQL

Используем функцию `EXTRACT` для возвращения значения года в формате `YYYY` и значения месяца в формате `MM` для указанных дат:

```

1 select mnth, mnth/12
2   from (
3 select ( extract(year from max_hd)
4        extract(year from min_hd) ) * 12
5        +
6        ( extract(month from max_hd)
7        extract(month from min_hd) ) as mnth
8   from (
9 select min(hiredate) as min_hd, max(hiredate) as max_hd
10  from emp
11        ) x
12        ) y

```

SQL Server

С помощью функции `DATEDIFF` определяем разницу между двумя датами, указывая в аргументе месяцы и годы, как формат возвращаемого результата[^]

```
1 select datediff(month,min_hd,max_hd),
2        datediff(year,min_hd,max_hd)
3    from (
4 select min(hiredate) min_hd, max(hiredate) max_hd
5    from emp
6        ) x
```

Обсуждение

DB2, MySQL и PostgreSQL

Решения для этих трех СУБД различаются только способом извлечения значений `MIN_HD` и `MAX_HD` в решении для PostgreSQL. Вычисление количества месяцев и лет между этими двумя датами реализуется одинаково для всех трех СУБД.

Вложенный запрос `X` возвращает предельные значения столбца `HIREDATE` таблицы `EMP`:

```
select min(hiredate) as min_hd,
       max(hiredate) as max_hd
   from emp
```

```
MIN_HD      MAX_HD
-----
17-DEC-1980 12-JAN-1983
```

Количество месяцев между значениями `MAX_HD` и `MIN_HD` вычисляется умножением на 12 разницы в годах между этими значениями, а затем добавлением к полученному результату разницы в месяцах между этим двумя значениями. Чтобы разобраться, как это работает, рассмотрим возвращение каждой даты по составляющим — году и месяцу:

```
select year(max_hd) as max_yr, year(min_hd) as min_yr,
       month(max_hd) as max_mon, month(min_hd) as min_mon
   from (
select min(hiredate) as min_hd, max(hiredate) as max_hd
   from emp
       ) x
```

```
MAX_YR      MIN_YR      MAX_MON      MIN_MON
-----
1983        1980            1            12
```

Количество месяцев между `MAX_HD` и `MIN_HD` по этим результатам вычисляется по простой формуле: $(1983 - 1980) \times 12 + (1 - 12)$. А количество лет между этими датами вычисляется делением полученного количества месяцев на 12. Опять же,

в зависимости от требуемого формата ответа, значение лет нужно округлить в большую или меньшую сторону.

Oracle и SQL Server

Вложенный запрос X возвращает предельные значения столбца HIREDATE таблицы EMP:

```
select min(hiredate) as min_hd, max(hiredate) as max_hd
       from emp
```

```
MIN_HD      MAX_HD
-----
17-DEC-1980 12-JAN-1983
```

Количество месяцев между указанными датами возвращается встроенными функциями Oracle и SQL Server (MONTHS_BETWEEN и DATEDIFF, соответственно). А количество лет между этими датами вычисляется делением полученного количества месяцев на 12.

8.5. Вычисление количества секунд, минут и часов между двумя датами

ЗАДАЧА

Требуется вычислить разницу в секундах, минутах и часах между двумя датами — например, между датами приема на работу HIREDATE служащих ALLEN и WARD.

РЕШЕНИЕ

Вычислив количество дней между указанными датами, можно с легкостью узнать количество секунд, минут и часов между ними, поскольку это единицы времени, составляющие день.

DB2

Разницу в днях между датами ALLEN_HD и WARD_HD определяем с помощью функции DAY. Секунды, минуты и часы вычисляем, умножая количество дней на соответствующие коэффициенты:

```
1 select dy*24 hr, dy*24*60 min, dy*24*60*60 sec
2       from (
3 select ( days(max(case when ename = 'WARD'
4                   then hiredate
5                   end)) -
6       days(max(case when ename = 'ALLEN'
7                   then hiredate
8                   end))
9       )as dy
```



```

10     from emp
11     ) x

```

MySQL

Разницу в днях между датами ALLEN_HD и WARD_HD определяем с помощью функции DATEDIFF. Секунды, минуты и часы вычисляем, умножая количество дней на соответствующие коэффициенты:

```

1 select datediff(day,allen_hd,ward_hd)*24 hr,
2        datediff(day,allen_hd,ward_hd)*24*60 min,
3        datediff(day,allen_hd,ward_hd)*24*60*60 sec
4     from (
5 select max(case when ename = 'WARD'
6            then hiredate
7            end) as ward_hd,
8        max(case when ename = 'ALLEN'
9            then hiredate
10           end) as allen_hd
11     from emp
12     ) x

```

SQL Server

Разницу в днях между датами ALLEN_HD и WARD_HD определяем с помощью функции DATEDIFF. Затем с помощью аргумента DATEPART указываем требуемые единицы времени:

```

1 select datediff(day,allen_hd,ward_hd,hour) as hr,
2        datediff(day,allen_hd,ward_hd,minute) as min,
3        datediff(day,allen_hd,ward_hd,second) as sec
4     from (
5 select max(case when ename = 'WARD'
6            then hiredate
7            end) as ward_hd,
8        max(case when ename = 'ALLEN'
9            then hiredate
10           end) as allen_hd
11     from emp
12     ) x

```

Oracle и PostgreSQL

Разницу в днях между датами ALLEN_HD и WARD_HD определяем с помощью операции вычитания. Секунды, минуты и часы вычисляем, умножая количество дней на соответствующие коэффициенты:

```

1 select dy*24 as hr, dy*24*60 as min, dy*24*60*60 as sec
2     from (
3 select (max(case when ename = 'WARD'
4            then hiredate
5            end) -

```

```

6      max(case when ename = 'ALLEN'
7          then hiredate
8          end)) as dy
9      from emp
10     ) x

```

Обсуждение

Значения столбца HIREDATE для служащих WARD и ALLEN возвращаются посредством вложенного запроса X:

```

select max(case when ename = 'WARD'
               then hiredate
               end) as ward_hd,
       max(case when ename = 'ALLEN'
               then hiredate
               end) as allen_hd
from emp

```

```

WARD_HD      ALLEN_HD
-----
22-FEB-1981  20-FEB-1981

```

Умножая полученное количество дней на 24 (часов в сутках), 1440 (минут в сутках) и 86 400 (секунд в сутках), получим соответствующие значения часов, минут и секунд.

8.6. Вычисление повторений каждого дня недели в году

ЗАДАЧА

Требуется вычислить количество повторений каждого дня недели в году.

РЕШЕНИЕ

Алгоритм решения этой задачи следующий:

1. Генерируем все возможные даты года.
2. Форматируем даты таким образом, чтобы они содержали соответствующий день недели.
3. Подсчитываем количество вхождений каждого дня недели.

DB2

Используем рекурсивное выражение WITH, чтобы избежать необходимости выполнять оператор SELECT для таблицы, содержащей не менее чем 366 строк. С помощью функции DAYNAME получаем название дня недели для каждой даты, а затем подсчитываем количество вхождений каждого из них:

```

1 with x (start_date,end_date)
2 as (
3 select start_date,
4         start_date + 1 year end_date
5    from (
6 select (current_date
7         dayofyear(current_date) day)
8         +1 day as start_date
9    from t1
10       ) tmp
11  union all
12 select start_date + 1 day, end_date
13    from x
14   where start_date + 1 day < end_date
15 )
16 select dayname(start_date),count(*)
17    from x
18   group by dayname(start_date)

```

MySQL

Сначала выполняем запрос по таблице T500, чтобы создать достаточное количество строк для возвращения каждого дня года. Затем с помощью функции DATE_FORMAT получаем название дня недели для каждой даты и подсчитываем количество вхождений каждого из них:

```

1 select date_format(
2         date_add(
3         cast(
4         concat(year(current_date),'-01-01')
5             as date),
6         interval t500.id-1 day),
7         '%W') day,
8         count(*)
9    from t500
10   where t500.id <= datediff(
11         cast(
12         concat(year(current_date)+1,'-01-01')
13             as date),
14         cast(
15         concat(year(current_date),'-01-01')
16             as date))
17  group by date_format(
18         date_add(
19         cast(
20         concat(year(current_date),'-01-01')
21             as date),
22         interval t500.id-1 day),
23         '%W')

```

Oracle

Для получения всех дней года используем рекурсивный оператор `CONNECT BY`:

```

1 with x as (
2 select level lvl
3     from dual
4     connect by level <= (
5         add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
6     )
7 )
8 select to_char(trunc(sysdate,'y')+lvl-1,'DAY'), count(*)
9     from x
10    group by to_char(trunc(sysdate,'y')+lvl-1,'DAY')
```

PostgreSQL

С помощью встроенной функции `GENERATE_SERIES` создаем по одной строке для каждого дня года. Затем с помощью функции `TO_CHAR` получаем название дня недели для каждой даты. Наконец, подсчитываем количество вхождений каждого дня недели:

```

1 select to_char(
2     cast(
3     date_trunc('year',current_date)
4     as date) + gs.id-1,'DAY'),
5     count(*)
6     from generate_series(1,366) gs(id)
7     where gs.id <= (cast
8         ( date_trunc('year',current_date) +
9         interval '12 month' as date) -
10    cast(date_trunc('year',current_date)
11        as date))
12    group by to_char(
13        cast(
14        date_trunc('year',current_date)
15        as date) + gs.id-1,'DAY')
```

SQL Server

Используем рекурсивное выражение `WITH`, чтобы избежать необходимости выполнять оператор `SELECT` для таблицы, содержащей не менее чем 366 строк. С помощью функции `DATENAME` получаем название дня недели для каждой даты, а затем подсчитываем количество вхождений каждого из них:

```

1 with x (start_date,end_date)
2 as (
3 select start_date,
4     dateadd(year,1,start_date) end_date
```

```

5     from (
6 select cast(
7     cast(year(getdate()) as varchar) + '-01-01'
8     as datetime) start_date
9     from t1
10    ) tmp
11 union all
12 select dateadd(day,1,start_date), end_date
13     from x
14  where dateadd(day,1,start_date) < end_date
15 )
16 select datename(dw,start_date),count(*)
17     from x
18  group by datename(dw,start_date)
19 OPTION (MAXRECURSION 366)

```

Обсуждение

DB2

Как показано далее, вложенный запрос в рекурсивном представлении X возвращает первый день текущего года:

```

select (current_date
       dayofyear(current_date) day)
       +1 day as start_date
from t1

```

```

START_DATE
-----
01-JAN-2005

```

На следующем шаге добавляем один год к дате START_DATE, чтобы получить начальную и конечную даты, которые требуется для генерирования каждой даты года. Далее приводится запрос для получения начальной даты START_DATE и конечной даты END_DATE и результаты его исполнения:

```

select start_date,
       start_date + 1 year end_date
from (
select (current_date
       dayofyear(current_date) day)
       +1 day as start_date
from t1
) tmp

```

```

START_DATE  END_DATE
-----
01-JAN-2005 01-JAN-2006

```

На следующем шаге рекурсивно инкрементируем дату `START_DATE` по одному дню, пока не получим значение меньше, чем конечная дата `END_DATE`. Далее показана часть строк, возвращаемых рекурсивным запросом X:

```
with x (start_date,end_date)
as (
select start_date,
       start_date + 1 year end_date
  from (
select (current_date -
       dayofyear(current_date) day)
       +1 day as start_date
  from t1
       ) tmp
 union all
select start_date + 1 day, end_date
  from x
 where start_date + 1 day < end_date
 )
select * from x
```

```
START_DATE END_DATE
-----
01-JAN-2005 01-JAN-2006
02-JAN-2005 01-JAN-2006
03-JAN-2005 01-JAN-2006
...
29-JAN-2005 01-JAN-2006
30-JAN-2005 01-JAN-2006
31-JAN-2005 01-JAN-2006
...
01-DEC-2005 01-JAN-2006
02-DEC-2005 01-JAN-2006
03-DEC-2005 01-JAN-2006
...
29-DEC-2005 01-JAN-2006
30-DEC-2005 01-JAN-2006
31-DEC-2005 01-JAN-2006
```

В завершение применяем функцию `DAYNAME` к строкам, возвращенным рекурсивным запросом X, для подсчета количества повторений каждого дня недели. Далее приведены соответствующий запрос и его результаты:

```
with x (start_date,end_date)
as (
select start_date,
       start_date + 1 year end_date
```

```

    from (
select (
    current_date -
    dayofyear(current_date) day)
    +1 day as start_date
    from t1
    ) tmp
union all
select start_date + 1 day, end_date
    from x
    where start_date + 1 day < end_date
)
select dayname(start_date),count(*)
    from x
    group by dayname(start_date)

```

```

START_DATE    COUNT(*)
-----
FRIDAY        52
MONDAY        52
SATURDAY      53
SUNDAY        52
THURSDAY     52
TUESDAY       52
WEDNESDAY    52

```

MySQL

Сначала выполняем запрос по таблице T500, чтобы создать по одной строке для каждого дня года. Команда в строке 4 возвращает первый день текущего года. Она делает это, добавляя к году даты, предоставленной функцией CURRENT_DATE, значений месяца и дня (создавая дату в формате дат MySQL по умолчанию). Соответствующий запрос и его результаты показаны далее:

```

select concat(year(current_date), '-01-01')
    from t1

```

```

START_DATE
-----
01-JAN-2005

```

Имея первый день текущего года, с помощью функции DATEADD последовательно добавляем к нему все значения из столбца T500.ID, создавая дату каждого дня года. Название дня для каждой даты получаем с помощью функции DATE_FORMAT. Чтобы сгенерировать требуемое количество строк из таблицы T500, определяем разницу в днях между первым днем текущего года и первым днем следующего года и возвращаем это количество строк (которое будет или 365 или 366). Соответствующий запрос и часть его результатов показаны далее:

```

select date_format(
    date_add(
        cast(
            concat(year(current_date), '-01-01')
                as date),
        interval t500.id-1 day),
        '%W') day
from t500
where t500.id <= datediff(
    cast(
        concat(year(current_date)+1, '-01-01')
            as date),
    cast(
        concat(year(current_date), '-01-01')
            as date))

```

```

DAY
-----
01-JAN-2005
02-JAN-2005
03-JAN-2005
...
29-JAN-2005
30-JAN-2005
31-JAN-2005
...
01-DEC-2005
02-DEC-2005
03-DEC-2005
...
29-DEC-2005
30-DEC-2005
31-DEC-2005

```

Получив все дни текущего года, подсчитываем количество вхождений каждого дня недели, возвращаемого функцией DAYNAME. Соответствующий запрос и его результаты показаны далее:

```

select date_format(
    date_add(
        cast(
            concat(year(current_date), '-01-01')
                as date),
        interval t500.id-1 day),
        '%W') day,
    count(*)
from t500

```



```

where t500.id <= datediff(
    cast(
        concat(year(current_date)+1, '-01-01')
        as date),
    cast(
        concat(year(current_date), '-01-01')
        as date))

group by date_format(
    date_add(
        cast(
            concat(year(current_date), '-01-01')
            as date),
        interval t500.id-1 day),
    '%W')

```

DAY	COUNT(*)
FRIDAY	52
MONDAY	52
SATURDAY	53
SUNDAY	52
THURSDAY	52
TUESDAY	52
WEDNESDAY	52

Oracle

В предоставленном решении строки для каждого дня текущего года можно сгенерировать или посредством выборки по сводной таблице T500 или посредством рекурсивных операторов CONNECT BY и WITH. Вызвав функцию TRUNC, усекаем текущую дату до первого дня текущего года.

Для решения с CONNECT BY/WITH сгенерировать последовательные числа, начиная с единицы, можно, используя псевдостолбец LEVEL. Чтобы сгенерировать требуемое количество строк из таблицы, фильтруем столбец ROWNUM или LEVEL по разнице в днях между первым днем текущего года и первым днем следующего года (которая будет или 365, или 366). На следующем шаге инкрементируем каждый день, добавляя значение ROWNUM или LEVEL к первому дню текущего года. Далее приводится соответствующий запрос:

```

/* Oracle 9i и более поздние версии */
with x as (
select level lvl
    from dual
    connect by level <= (
        add_months(trunc(sysdate, 'y'), 12) - trunc(sysdate, 'y')
    )
)
select trunc(sysdate, 'y') + lvl - 1 from x

```

В случае решения со сводной таблицей можно использовать любую таблицу или представление, содержащее как минимум 366 строк. Поскольку Oracle поддерживает функцию `ROWNUM`, можно обойтись без таблицы со списком инкрементирующихся значений, начинающихся с 1. Рассмотрим следующий пример с использованием сводной таблицы `T500` для формирования списка всех дат текущего года:

```
/* Oracle 8i и более ранние версии */
select trunc(sysdate,'y')+rownum-1 start_date
       from t500
       where rownum <= (add_months(trunc(sysdate,'y'),12)
                        - trunc(sysdate,'y'))
```

```
START_DATE
-----
01-JAN-2005
02-JAN-2005
03-JAN-2005
...
29-JAN-2005
30-JAN-2005
31-JAN-2005
...
01-DEC-2005
02-DEC-2005
03-DEC-2005
...
29-DEC-2005
30-DEC-2005
31-DEC-2005
```

Независимо от выбранного подхода, в конечном итоге нам нужно использовать функцию `TO_CHAR`, чтобы получить название дня недели для каждой даты, а затем подсчитать количество вхождений каждого из них. Далее приводятся соответствующие запросы и конечные результаты:

```
/* Oracle 9i и более поздние версии */
with x as (
select level lvl
       from dual
       connect by level <= (
          add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
        )
)
select to_char(trunc(sysdate,'y')+lvl-1,'DAY'), count(*)
       from x
       group by to_char(trunc(sysdate,'y')+lvl-1,'DAY')
```

```

/* Oracle 8i и более ранние версии */
select to_char(trunc(sysdate,'y')+rownum-1,'DAY') start_date,
       count(*)
  from t500
 where rownum <= (add_months(trunc(sysdate,'y'),12)
                  - trunc(sysdate,'y'))
 group by to_char(trunc(sysdate,'y')+rownum-1,'DAY')

```

```

START_DATE  COUNT(*)
-----
FRIDAY      52
MONDAY      52
SATURDAY    53
SUNDAY      52
THURSDAY    52
TUESDAY     52
WEDNESDAY   52

```

PostgreSQL

На первом шаге используем функцию `DATE_TRUNC`, чтобы получить год текущей даты (как показано далее, выборка делается по таблице `T1`, поэтому возвращается только одна строка):

```

select cast(
       date_trunc('year',current_date)
       as date) as start_date
  from t1

```

```

START_DATE
-----
01-JAN-2005

```

Затем выполняем выборку по источнику строк (по сути, по любому табличному выражению), содержащему как минимум 366 строк. В приведенном решении источником строк служит функция `GENERATE_SERIES`. Но, конечно же, вместо нее можно использовать таблицу `T500`. Затем инкрементируем первый день текущего года по одному дню, пока не получим все даты года. Далее приводится соответствующий запрос и его частичные результаты:

```

select cast( date_trunc('year',current_date)
           as date) + gs.id-1 as start_date
  from generate_series (1,366) gs(id)
 where gs.id <= (cast
                ( date_trunc('year',current_date) +
                  interval '12 month' as date) -
                cast(date_trunc('year',current_date)
                  as date))

```

```

START_DATE
-----
01-JAN-2005
02-JAN-2005
03-JAN-2005
...
29-JAN-2005
30-JAN-2005
31-JAN-2005
...
01-DEC-2005
02-DEC-2005
03-DEC-2005
...
29-DEC-2005
30-DEC-2005
31-DEC-2005

```

В завершение используем функцию `TO_CHAR`, чтобы получить название дня недели для каждой даты, а затем подсчитать количество вхождений каждого из них. Далее приводятся соответствующий запрос и конечные результаты:

```

select to_char(
        cast(
            date_trunc('year',current_date)
                as date) + gs.id-1,'DAY') as start_dates,
        count(*)
    from generate_series(1,366) gs(id)
 where gs.id <= (cast
                ( date_trunc('year',current_date) +
                  interval '12 month' as date) -
                cast(date_trunc('year',current_date)
                    as date))
 group by to_char(
        cast(
            date_trunc('year',current_date)
                as date) + gs.id-1,'DAY')

```

```

START_DATE  COUNT(*)
-----
FRIDAY      52
MONDAY      52
SATURDAY    53
SUNDAY      52
THURSDAY    52
TUESDAY     52
WEDNESDAY   52

```

SQL Server

Как показано далее, вложенный запрос в рекурсивном представлении X возвращает первый день текущего года:

```
select cast(
cast(year(getdate()) as varchar) + '-01-01'
as datetime) start_date
from t1
```

```
START_DATE
-----
01-JAN-2005
```

Получив первый день `START_DAY` текущего года, добавляем к нему один год, чтобы получить конечный день `END_DAY`. Обе эти даты требуется для генерирования каждой даты года. Далее приводится запрос для получения начальной даты `START_DATE` и конечной даты `END_DATE` и результаты его исполнения:

```
select start_date,
       dateadd(year,1,start_date) end_date
from (
select cast(
       cast(year(getdate()) as varchar) + '-01-01'
       as datetime) start_date
from t1
) tmp
```

```
START_DATE  END_DATE
-----
01-JAN-2005 01-JAN-2006
```

На следующем шаге рекурсивно инкрементируем дату `START_DATE` по одному дню, пока не получим значение на один день меньше, чем конечная дата `END_DATE`. Далее приводятся соответствующий рекурсивный запрос X и часть возвращаемых им строк:

```
with x (start_date,end_date)
as (
select start_date,
       dateadd(year,1,start_date) end_date
from (
select cast(
       cast(year(getdate()) as varchar) + '-01-01'
       as datetime) start_date
from t1
) tmp
union all
select dateadd(day,1,start_date), end_date
from x
```

```

where dateadd(day,1,start_date) < end_date
)
select * from x
OPTION (MAXRECURSION 366)

```

```

START_DATE  END_DATE
-----
01-JAN-2005 01-JAN-2006
02-JAN-2005 01-JAN-2006
03-JAN-2005 01-JAN-2006
...
29-JAN-2005 01-JAN-2006
30-JAN-2005 01-JAN-2006
31-JAN-2005 01-JAN-2006
...
01-DEC-2005 01-JAN-2006
02-DEC-2005 01-JAN-2006
03-DEC-2005 01-JAN-2006
...
29-DEC-2005 01-JAN-2006
30-DEC-2005 01-JAN-2006
31-DEC-2005 01-JAN-2006

```

В завершение применяем функцию DATENAME к строкам, возвращенным рекурсивным запросом X, и подсчитываем количество повторений каждого дня недели:

```

with x(start_date,end_date)
as (
select start_date,
       dateadd(year,1,start_date) end_date
from (
select cast(
       cast(year(getdate()) as varchar) + '-01-01'
       as datetime) start_date
from t1
) tmp
union all
select dateadd(day,1,start_date), end_date
from x
where dateadd(day,1,start_date) < end_date
)
select datename(dw,start_date), count(*)
from x
group by datename(dw,start_date)
OPTION (MAXRECURSION 366)

```

```

START_DATE COUNT(*)
-----
FRIDAY 52
MONDAY 52

```

```
SATURDAY 53
SUNDAY 52
THURSDAY 52
TUESDAY 52
WEDNESDAY 52
```

8.7. Вычисление разницы в днях между датами двух записей

ЗАДАЧА

Требуется вычислить разницу в днях между двумя датами, хранящимися в разных строках таблицы. Например, для каждого служащего отдела 10 нужно вычислить количество дней между датой его приема на работу и датой приема на работу следующего служащего (который может быть в другом отделе).

РЕШЕНИЕ

Ключ к решению этой задачи — определить ближайшую дату `HIREDATE` после даты приема на работу текущего служащего. Затем просто воспользоваться методом из *рецепта 8.2*, чтобы вычислить разницу в днях между этими датами.

DB2

Используем скалярный подзапрос для нахождения значения `HIREDATE`, следующего за текущим. Затем с помощью функции `DAYS` вычисляем разницу в днях между этими двумя датами:

```
1 select x.*,
2     days(x.next_hd) - days(x.hiredate) diff
3     from (
4 select e.deptno, e.ename, e.hiredate,
5     lead(hiredate)over(order by hiredate) next_hd
6     from emp e
7     where e.deptno = 10
8     ) x
```

MySQL и SQL Server

Используем функцию `LEAD` для нахождения следующей строки. Затем с помощью функции `DATEDIFF` вычисляем разницу между ними в днях. Далее показано использование варианта функции `DATEDIFF` для SQL Server:

```
1 select x.ename, x.hiredate, x.next_hd,
2     datediff(x.hiredate,x.next_hd,day) as diff
3     from (
4 select deptno, ename, hiredate,
5     lead(hiredate)over(order by hiredate) as next_hd
```

```

6   from emp e
7     ) x
8   where e.deptno=10

```

Для MySQL можно не использовать первый аргумент (`day`), а также поменять местами два других аргумента:

```
2 datediff(x.next_hd, x.hiredate) diff
```

Oracle

Используем оконную функцию `LEAD OVER`, чтобы определить следующее значение `HIREDATE` относительно текущего:

```

1 select ename, hiredate, next_hd,
2        next_hd - hiredate diff
3   from (
4 select deptno, ename, hiredate,
5        lead(hiredate)over(order by hiredate) next_hd
6   from emp
7     )
8  where deptno=10

```

PostgreSQL

Используем скалярный подзапрос для нахождения значения `HIREDATE`, следующего за текущим. Затем просто вычитаем меньшее значение из большего:

```

1 select x.*,
2        x.next_hd - x.hiredate as diff
3   from (
4 select e.deptno, e.ename, e.hiredate,
5        lead(hiredate)over(order by hiredate) as next_hd
6   from emp e
7  where e.deptno = 10
8     ) x

```

Обсуждение

Несмотря на различия в синтаксисе, во всех приведенных решениях используется одинаковый подход: сначала определяется строка со следующей датой посредством оконной функции `LEAD`, а затем вычисляется разница между двумя датами с помощью метода, описанного в *рецепте 8.2*.

Возможность обращения к строкам, соседствующим с текущей строкой, не прибегая к дополнительным операциям объединения, позволяет создавать более читабельный и эффективный код. При работе с оконными функциями следует помнить, что они выполняются после предиката `WHERE`, поэтому в решении необходимо использовать вложенный запрос. Перемещение фильтра по столбцу `DEPTNO` во вложенный запрос изменило бы результаты — в частности, обрабатывались бы значения

HIREDATE только для отдела 10. Важно отметить особое поведение функций LEAD и LAG СУБД Oracle при наличии дубликатов. В предисловии упоминалось, что код приводимых рецептов не содержит никаких мер предосторожности, поскольку невозможно предвидеть то огромное количество ситуаций, которые могут вызвать сбой кода. Кроме этого, даже если и можно было бы предвидеть все возможные проблемы, решения с кодом защиты от них были бы, скорее всего, громоздкими и трудно понимаемыми. Поэтому в большинстве случаев целью рецепта является представление метода решения конкретной задачи, который можно использовать в реальной системе, но который необходимо всесторонне протестировать и многократно откорректировать для работы с конкретными используемыми данными. Но в рассматриваемом случае есть один момент, который необходимо обсудить просто по той причине, что обходное решение может быть не совсем очевидным, особенно для пользователей СУБД иных, чем Oracle. Как вы могли заметить, таблица EMP не содержит дубликатов значений столбца HIREDATE, но ситуация с наличием дубликатов значений данных, безусловно, возможна (и даже с большой вероятностью). Рассмотрим даты приема на работу HIREDATE служащих отдела DEPTNO 10:

```
select ename, hiredate
       from emp
      where deptno=10
      order by 2
```

```
ENAME  HIREDATE
-----  -
CLARK  09-JUN-1981
KING   17-NOV-1981
MILLER 23-JAN-1982
```

Для целей нашего примера добавим в таблицу четыре дубликата, чтобы было пять служащих (включая служащего KING), принятых на работу 17 ноября.

```
insert into emp (empno,ename,deptno,hiredate)
values (1,'ant',10,to_date('17-NOV-1981'))
```

```
insert into emp (empno,ename,deptno,hiredate)
values (2,'joe',10,to_date('17-NOV-1981'))
```

```
insert into emp (empno,ename,deptno,hiredate)
values (3,'jim',10,to_date('17-NOV-1981'))
```

```
insert into emp (empno,ename,deptno,hiredate)
values (4,'choi',10,to_date('17-NOV-1981'))
```

```
select ename, hiredate
       from emp
      where deptno=10
      order by 2
```

```

ENAME HIREDATE
-----
CLARK 09-JUN-1981
ant 17-NOV-1981
joe 17-NOV-1981
KING 17-NOV-1981
jim 17-NOV-1981
choi 17-NOV-1981
MILLER 23-JAN-1982

```

Теперь отдел 10 содержит несколько служащих с одинаковой датой приема на работу. Если попытаться применить предлагаемое решение (перенести фильтр во внутренний запрос, чтобы работать со значениями HIREDATE только сотрудников отдела 10) с этим результирующим множеством, то получим следующий результат:

```

select ename, hiredate, next_hd,
       next_hd - hiredate diff
   from (
select deptno, ename, hiredate,
       lead(hiredate)over(order by hiredate) next_hd
   from emp
  where deptno=10
   )

```

ENAME	HIREDATE	NEXT_HD	DIFF
CLARK	09-JUN-1981	17-NOV-1981	161
ant	17-NOV-1981	17-NOV-1981	0
joe	17-NOV-1981	17-NOV-1981	0
KING	17-NOV-1981	17-NOV-1981	0
jim	17-NOV-1981	17-NOV-1981	0
choi	17-NOV-1981	23-JAN-1982	67
MILLER	23-JAN-1982	(null)	(null)

Как можно видеть, значения DIFF для четырех из пяти служащих, принятых на работу в один и тот же день, равны нулю. Это неправильно. Дата приема на работу всех служащих, принятых на работу в один день, должна сравниваться со значением HIREDATE, следующим за датой их приема на работу. Иными словами, дата приема на работу всех служащих, принятых 17 ноября, должна сравниваться с датой HIREDATE служащего MILLER. Но здесь возникает проблема, состоящая в том, что функция LEAD упорядочивает строки по столбцу HIREDATE, но не пропускает дубликатов значений этого столбца. Поэтому результат сравнения значения HIREDATE, например, служащего ANT со значением HIREDATE служащего JOE будет ноль, отсюда и нулевое значение столбца DIFF для служащего ANT. К счастью, Oracle обеспечивает простое обходное решение для подобных проблем: функции LEAD можно передавать аргумент, указывающий точное местонахождение следующей строки (т. е. следующая строка, десятая строка и т. д.). Таким образом, для служащего ANT его значение HIREDATE сравнивается не со следующей строкой, а с пятой (перескакиваем через все другие дубликаты), т. е. со строкой служащего MILLER. Подобным образом,

служащий JOE отстоит на четыре строки от служащего MILLER, JIM — на три, KING — на две, а CHOI — на одну, т. е. находится в строке, предшествующей MILLER. Чтобы получить правильный ответ, просто передаем функции LEAD в аргументах расстояние от каждого сотрудника до сотрудника MILLER:

```
select ename, hiredate, next_hd,
       next_hd - hiredate diff
  from (
select deptno, ename, hiredate,
       lead(hiredate,cnt-rn+1)over(order by hiredate) next_hd
  from (
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
  from emp
 where deptno=10
  )
  )
```

ENAME	HIREDATE	NEXT_HD	DIFF
CLARK	09-JUN-1981	17-NOV-1981	161
ant	17-NOV-1981	23-JAN-1982	67
joe	17-NOV-1981	23-JAN-1982	67
jim	17-NOV-1981	23-JAN-1982	67
choi	17-NOV-1981	23-JAN-1982	67
KING	17-NOV-1981	23-JAN-1982	67
MILLER	23-JAN-1982	(null)	(null)

Теперь результат правильный. Значение HIREDATE всех служащих, принятых на работу в один день, сравнивается со следующим по величине значением HIREDATE, а не с совпадающим значением в следующей строке. Чтобы разобраться с работой откорректированного решения, просто разобьем запрос на части.

Сначала рассмотрим вложенный запрос:

```
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
  from emp
 where deptno=10
```

DEPTNO	ENAME	HIREDATE	CNT	RN
10	CLARK	09-JUN-1981	1	1
10	ant	17-NOV-1981	5	1
10	joe	17-NOV-1981	5	2
10	jim	17-NOV-1981	5	3
10	choi	17-NOV-1981	5	4
10	KING	17-NOV-1981	5	5
10	MILLER	23-JAN-1982	1	1

Оконная функция `COUNT OVER` подсчитывает количество вхождений каждого значения `HIREDATE` и возвращает это значение в каждую строку. Для каждого дубликата значения `HIREDATE` в каждую строку с этим значением возвращается значение 5. Оконная функция `ROW_NUMBER OVER` ранжирует всех служащих по их `EMPNO`. Ранги делятся по `HIREDATE`, поэтому все служащие, для которых отсутствуют дубликаты `HIREDATE`, получают ранг 1. Ранг можно использовать, как расстояние до следующего большего значения `HIREDATE` (т. е. `HIREDATE` служащего `MILLER`). Для этого при вызове функции `LEAD` для каждой строки значение ранга `RN` вычитается из счетчика дубликатов `CNT` и к результату добавляется 1:

```
select deptno, ename, hiredate,
       cnt-rn+1 distance_to_miller,
       lead(hiredate,cnt-rn+1)over(order by hiredate) next_hd
from (
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
from emp
where deptno=10
)
```

DEPTNO	ENAME	HIREDATE	DISTANCE_TO_MILLER	NEXT_HD
10	CLARK	09-JUN-1981	1	17-NOV-1981
10	ant	17-NOV-1981	5	23-JAN-1982
10	joe	17-NOV-1981	4	23-JAN-1982
10	jim	17-NOV-1981	3	23-JAN-1982
10	choi	17-NOV-1981	2	23-JAN-1982
10	KING	17-NOV-1981	1	23-JAN-1982
10	MILLER	23-JAN-1982	1	(null)

Как можно видеть, передача функции `LEAD` соответствующего расстояния до строки сравнения позволяет ей выполнять вычитание с правильными датами.

8.8. Подведем итоги

Несмотря на свою распространенность, типы дат обладают своими особенностями вследствие их более сложной структурированности. Кроме того, обработка этих особенностей в разных СУБД менее стандартизована, чем многие другие области, но каждая СУБД содержит основной набор функций, выполняющих сходные задачи, даже если соответствующий синтаксис может слегка различаться. Овладение этим набором функций обеспечит вам успешную работу с датами.

Работа с датами

В этой главе рассматриваются рецепты для поиска и модифицирования данных. В запросах очень часто приходится работать с датами. Поэтому нужно хорошо понимать тонкости такой работы и разобраться с функциями, предоставляемыми для этого используемой СУБД. Рецепты в этой главе формируют важную основу для будущей работы с более сложными запросами, содержащими не только даты, но и значения времени.

Прежде чем приступать к рассмотрению рецептов, мы хотим еще раз подчеркнуть главную идею этой книги (уже сформулированную в предисловии): ее рецепты лишь обозначают вам подходы к решению конкретных задач. Рассматривая проведенные в книге рецепты, попробуйте представить себе более общую стратегию их применения. Например, если рецепт решает задачу для текущего месяца, учтите, что с небольшими изменениями его, скорее всего, можно использовать для любого месяца. Опять же, эти рецепты предоставляют всего лишь общие направления, а не абсолютно конечные решения. В рамках этой книги невозможно предоставить ответы на все возможные задачи, но, понимая изложенный в ней материал, вы сможете с легкостью модифицировать ее рецепты для решения своих конкретных проблем. Учитывайте также возможность существования альтернативных версий решений. Например, если в решении используется определенная функция, предоставляемая вашей СУБД, имеет смысл потратить немного времени и усилий на поиск альтернативного варианта, который может быть более или менее эффективным, чем исходное решение. Знание доступных опций позволит вам улучшить свои навыки программирования SQL.



В рецептах этой главы используются простые типы дат. Если вы работаете с более сложными типами дат, вам нужно будет откорректировать приведенные здесь решения соответствующим образом.

9.1. Определение високосного года

ЗАДАЧА

Требуется определить, является ли текущий год високосным.

РЕШЕНИЕ

Несомненно, читатели с некоторым опытом работы с SQL знакомы с несколькими методами решения этой задачи. Практически все решения, которые известны нам,

работают достаточно хорошо, но рассматриваемое в этом рецепте решение является, наверное, наиболее простым. Это решение просто проверяет последний день февраля, и если это 29-е число, то текущий год является високосным.

DB2

Используем рекурсивный оператор `WITH`, чтобы получить все дни февраля, а затем с помощью агрегатной функции `MAX` определяем последний из них:

```

1 with x (dy,mth)
2   as (
3   select dy, month(dy)
4     from (
5   select (current_date -
6           dayofyear(current_date) days +1 days)
7           +1 months as dy
8     from t1
9        ) tmp1
10  union all
11  select dy+1 days, mth
12    from x
13   where month(dy+1 day) = mth
14 )
15  select max(day(dy))
16    from x

```

Oracle

Для определения последнего числа февраля используем функцию `LAST_DAY`:

```

1 select to_char(
2         last_day(add_months(trunc(sysdate,'y'),1)),
3         'DD')
4   from t1

```

PostgreSQL

Используем функцию `GENERATE_SERIES`, чтобы получить все дни февраля, а затем с помощью агрегатной функции `MAX` определяем последний из них:

```

1 select max(to_char(tmp2.dy+x.id,'DD')) as dy
2   from (
3   select dy, to_char(dy,'MM') as mth
4     from (
5   select cast(cast(
6             date_trunc('year',current_date) as date)
7             + interval '1 month' as date) as dy
8     from t1
9        ) tmp1

```

```

10         ) tmp2, generate_series (0,29) x(id)
11 where to_char(tmp2.dy+x.id, 'MM') = tmp2.mth

```

MySQL

Для определения последнего числа февраля используем функцию `LAST_DAY`:

```

1 select day(
2     last_day(
3     date_add(
4     date_add(
5     date_add(current_date,
6         interval -dayofyear(current_date) day),
7         interval 1 day),
8         interval 1 month))) dy
9 from t1

```

SQL Server

Используем рекурсивный оператор `WITH`, чтобы получить все дни февраля, а затем с помощью агрегатной функции `MAX` определяем последний из них:

```

select coalesce
    (day
    (cast(concat
    (year(getdate()), '-02-29')
    as date))
    ,28);

```

Обсуждение

DB2

Вложенный запрос `tmp1` в рекурсивном представлении `X` возвращает первый день февраля следующим образом:

1. Определяем текущий день.
2. Посредством функции `DAYOFYEAR` определяем количество дней в текущем году, представляемом текущей датой.
3. Вычитаем полученное количество дней из текущей даты, чтобы получить 31 декабря предыдущего года, а затем добавляем единицу, чтобы получить 1 января текущего года.
4. Добавляем один месяц, чтобы получить 1 февраля.

Соответствующий запрос и конечный результат всех этих математических операций показан далее:

```

select (current_date
    dayofyear(current_date) days +1 days) +1 months as dy
from t1

```

```
DY
-----
01-FEB-2005
```

На следующем шаге получаем число месяца для даты, возвращенной вложенным запросом TMP1, используя для этого функцию MONTH:

```
select dy, month(dy) as mth
  from (
select (current_date
       dayofyear(current_date) days +1 days) +1 months as dy
  from t1
     ) tmp1
```

```
DY          MTH
-----
01-FEB-2005  2
```

К этому моменту мы получили начальную точку для рекурсивной операции, которая генерирует все дни февраля. Чтобы получить каждый день февраля, многократно добавляем один день к DY, пока не выйдем за пределы февраля. Далее приводится соответствующая операция WITH и часть возвращенных ею результатов:

```
with x (dy,mth)
  as (
select dy, month(dy)
  from (
select (current_date -
       dayofyear(current_date) days +1 days) +1 months as dy
  from t1
     ) tmp1
  union all
select dy+1 days, mth
  from x
  where month(dy+1 day) = mth
 )
select dy,mth
  from x
```

```
DY          MTH
-----
01-FEB-2005  2
...
10-FEB-2005  2
...
28-FEB-2005  2
```

В завершение применяем функцию MAX к столбцу DY, возвращая последнее число февраля. Если оно равно 29, то год високосный.

Oracle

На первом шаге определяем начальную дату года, используя для этого функцию TRUNC:

```
select trunc(sysdate,'y')
  from t1
```

```
DY
-----
01-JAN-2005
```

Поскольку первый день года — это первое января, на следующем шаге добавляем один месяц, чтобы получить первое февраля:

```
select add_months(trunc(sysdate,'y'),1) dy
  from t1
```

```
DY
-----
01-FEB-2005
```

Затем определяем последнее число февраля, используя для этого функцию LAST_DAY:

```
select last_day(add_months(trunc(sysdate,'y'),1)) dy
  from t1
```

```
DY
-----
28-FEB-2005
```

В завершение (по желанию) посредством функции TO_CHAR возвращаем 28 или 29.

PostgreSQL

На первом шаге применяем к результатам, возвращенным вложенным запросом TMP1, функцию DATE_TRUNC, чтобы определить начало текущего года, и приводим полученный результат к типу DATE:

```
select cast(date_trunc('year',current_date) as date) as dy
  from t1
```

```
DY
-----
01-JAN-2005
```

На следующем шаге добавляем один месяц к первому числу текущего года, чтобы получить первый день февраля, опять приводя полученный результат к типу DATE:

```
select cast(cast(
    date_trunc('year',current_date) as date)
    + interval '1 month' as date) as dy
  from t1
```

```
DY
-----
01-FEB-2005
```

Затем из вложенного запроса TMP1 возвращаем значение DY, применяя к которому функцию TO_CHAR получаем порядковый номер месяца:

```
select dy, to_char(dy,'MM') as mth
  from (
select cast(cast(
      date_trunc('year',current_date) as date)
      + interval '1 month' as date) as dy
  from t1
    ) tmp1
```

```
DY MTH
----- ---
01-FEB-2005 2
```

Полученные до сих пор результаты составляют результирующее множество вложенного запроса TMP2. После этого с помощью функции GENERATE_SERIES возвращаем 29 строк (значения с 1 по 29). Каждая возвращенная функцией GENERATE_SERIES строка (которые все вместе образуют множество под псевдонимом X) добавляется к значению DY из вложенного запроса TMP2. Далее приводятся соответствующий запрос и частичные результаты:

```
select tmp2.dy+x.id as dy, tmp2.mth
  from (
select dy, to_char(dy,'MM') as mth
  from (
select cast(cast(
      date_trunc('year',current_date) as date)
      + interval '1 month' as date) as dy
  from t1
    ) tmp1
    ) tmp2, generate_series (0,29) x(id)
where to_char(tmp2.dy+x.id,'MM') = tmp2.mth
```

```
DY          MTH
----- ---
01-FEB-2005 02
...
10-FEB-2005 02
...
28-FEB-2005 02
```

В завершение определяем последнее число февраля, используя для этого функцию MAX. Полученное значение обрабатываем функцией TO_CHAR, получая 28 или 29.

MySQL

На первом шаге с помощью функции `DATE_ADD` вычисляем первый день текущего года, вычитая из текущей даты соответствующее количество истекших дней и добавляя к результату единицу. Далее приводится соответствующий запрос и его результаты:

```
select date_add(
    date_add(current_date,
        interval -dayofyear(current_date) day),
        interval 1 day) dy
from t1
```

```
DY
-----
01-JAN-2005
```

Затем добавляем один месяц, используя функцию `DATE_ADD`:

```
select date_add(
    date_add(
        date_add(current_date,
            interval -dayofyear(current_date) day),
            interval 1 day),
        interval 1 month) dy
from t1
```

```
DY
-----
01-FEB-2005
```

Потом определяем последнее число февраля, используя для этого функцию `LAST_DAY`:

```
select last_day(
    date_add(
        date_add(
            date_add(current_date,
                interval -dayofyear(current_date) day),
                interval 1 day),
            interval 1 month)) dy
from t1
```

```
DY
-----
28-FEB-2005
```

В завершение (по желанию) с помощью функции `DAY` возвращаем 28 или 29.

SQL Server

В большинстве СУБД новую дату можно создать, создав строку в стандартном формате даты, а затем посредством функции `CAST` привести ее к типу данных `DATE`. Таким образом, можно использовать значение текущего года, получив его из текущей даты. Для SQL Server эта операция выполняется применением функции `YEAR` к результатам функции `GET_DATE`:

```
select YEAR(GETDATE());
```

В результате мы получим значение года в виде целочисленного значения. На основе этого значения можно создать число 29 февраля, используя для этого функции `CONCAT` и `CAST`:

```
select cast(concat
            (year(getdate()), '-02-29');
```

Но если текущий год не високосный, то полученный год будет недействительным. Например, числа 2019-02-29 не существует. В результате при попытке извлечь из него составные части посредством, например, оператора `DAY` будет возвращено значение `NULL`. Поэтому, чтобы определить наличие числа 29, нужно использовать функции `COALESCE` и `DAY`.

9.2. Определение количества дней в году

ЗАДАЧА

Требуется вычислить количество дней в текущем году.

РЕШЕНИЕ

Количество дней в текущем году — это разница в днях между первым днем следующего года и первым днем текущего года. Решение для каждой СУБД следует такому алгоритму:

1. Определяем первый день текущего года.
2. К полученной дате добавляем один год (чтобы получить первый день следующего года).
3. Извлекаем текущий год из результата шага 2.

Реализации этого алгоритма для конкретных СУБД различаются только встроенными функциями, используемыми для выполнения этих шагов.

DB2

С помощью функции `DAYOFYEAR` определяем первый день текущего года, а затем используем функцию `DAYS`, чтобы вычислить количество дней в текущем году:

```
1 select days((curr_year + 1 year)) - days(curr_year)
2   from (
```

```

3 select (current_date -
4         dayofyear(current_date) day +
5         1 day) curr_year
6   from t1
7         ) x

```

Oracle

С помощью функции TRUNC вычисляем первый день текущего года, а затем с помощью функции ADD_MONTHS вычисляем первый день следующего года:

```

1 select add_months(trunc(sysdate, 'y'), 12) - trunc(sysdate, 'y')
2   from dual

```

PostgreSQL

С помощью функции DATE_TRUNC вычисляем первый день текущего года, а затем с помощью интервальной арифметики определяем начало следующего года:

```

1 select cast((curr_year + interval '1 year') as date) - curr_year
2   from (
3 select cast(date_trunc('year', current_date) as date) as curr_year
4   from t1
5         ) x

```

MySQL

С помощью функции ADDDATE вычисляем первый день текущего года, а затем с помощью функции DATEDIFF и интервальной арифметики определяем количество дней в году:

```

1 select datediff((curr_year + interval 1 year), curr_year)
2   from (
3 select adddate(current_date, -dayofyear(current_date)+1) curr_year
4   from t1
5         ) x

```

SQL Server

С помощью функции DATEADD вычисляем первый день текущего года, а затем с помощью функции DATEDIFF возвращаем количество дней в текущем году:

```

1 select datediff(d, curr_year, dateadd(yy, 1, curr_year))
2   from (
3 select dateadd(d, -datepart(dy, getdate())+1, getdate()) curr_year
4   from t1
5         ) x

```

Обсуждение

DB2

На первом шаге определяем первый день текущего года. Для этого сначала с помощью функции `DAYOFYEAR` определяем количество дней, прошедших с начала текущего года. Далее вычитаем полученное значение из текущей даты, чтобы получить последний день предыдущего года, и добавляем к полученному значению 1:

```
select (current_date
       dayofyear(current_date) day +
       1 day) curr_year
from t1
```

```
CURR_YEAR
-----
01-JAN-2005
```

Получив первый день текущего года, просто добавляем к нему один год, в результате получая первый день следующего года. Наконец, вычитаем начало текущего года из начала следующего года.

Oracle

На первом шаге определяем первый день текущего года. Для этого вызываем встроенную функцию `TRUNC`, передавая ей в качестве второго аргумента английскую букву 'y' (что усекает результат, возвращая только начальную дату года):

```
select select trunc(sysdate,'y') curr_year
from dual
```

```
CURR_YEAR
-----
01-JAN-2005
```

К полученной дате добавляем один год, чтобы получить первый день следующего года. Наконец, вычитаем текущий год из следующего, получая количество дней в текущем году.

PostgreSQL

Начинаем с определения первого дня текущего года, вызывая функцию `DATE_TRUNC` следующим образом:

```
select cast(date_trunc('year',current_date) as date) as curr_year
from t1
```

```
CURR_YEAR
-----
01-JAN-2005
```

К полученной дате добавляем один год, чтобы получить первый день следующего года. Теперь осталось только вычесть дату текущего года из даты следующего, что и даст нам количество дней в текущем году.

MySQL

На первом шаге определяем первый день текущего года. Для этого сначала с помощью функции `DAYOFYEAR` определяем количество дней, прошедших с начала текущего года. Затем вычитаем полученное значение из текущей даты и добавляем 1:

```
select adddate(current_date, -dayofyear(current_date)+1) curr_year
from t1
```

```
CURR_YEAR
-----
01-JAN-2005
```

Получив первый день текущего года, просто добавляем к нему один год, в результате получая первый день следующего года. Наконец, вычитаем начало текущего года из начала следующего года, получая в результате количество дней в текущем году.

SQL Server

На первом шаге определяем первый день текущего года, для чего с помощью функций `DATEADD` и `DATPART` вычитаем из текущей даты количество дней, прошедших с начала года, и добавляем 1:

```
select dateadd(d, -datepart(dy, getdate())+1, getdate()) curr_year
from t1
```

```
CURR_YEAR
-----
01-JAN-2005
```

Получив первый день текущего года, просто добавляем к нему один год, в результате получая первый день следующего года. Наконец, вычитаем начало текущего года из начала следующего года, получая в результате количество дней в текущем году.

9.3. Извлечение из даты единиц времени

ЗАДАЧА

Требуется разбить текущую дату на шесть составляющих частей: день, месяц, год, секунду, минуту и час и вернуть результаты в виде чисел.

РЕШЕНИЕ

Этот рецепт применим не только к текущей дате, но и к любой произвольной дате. В настоящее время большинство рассматриваемых СУБД для извлечения частей даты реализуют стандартную ANSI-функцию `EXTRACT`. Исключением является только SQL Server. Все они также поддерживают собственные методы.

DB2

Эта СУБД реализует набор встроенных функций, позволяющих с легкостью извлекать части даты. Функции удобно называются по единицам времени, которые они возвращают: HOUR, MINUTE, SECOND, DAY, MONTH и YEAR. Далее приводится пример использования всех этих функций:

```

1 select hour( current_timestamp ) hr,
2 minute( current_timestamp ) min,
3 second( current_timestamp ) sec,
4 day( current_timestamp ) dy,
5 month( current_timestamp ) mth,
6 year( current_timestamp ) yr
7 from t1

```

```

select
    extract(hour from current_timestamp)
  , extract(minute from current_timestamp)
  , extract(second from current_timestamp)
  , extract(day from current_timestamp)
  , extract(month from current_timestamp)
  , extract(year from current_timestamp)

```

HR	MIN	SEC	DY	MTH	YR
20	28	36	15	6	2005

Oracle

Для извлечения единиц времени из даты используем функции TO_CHAR и TO_NUMBER:

```

1 select to_number(to_char(sysdate,'hh24')) hour,
2        to_number(to_char(sysdate,'mi')) min,
3        to_number(to_char(sysdate,'ss')) sec,
4        to_number(to_char(sysdate,'dd')) day,
5        to_number(to_char(sysdate,'mm')) mth,
6        to_number(to_char(sysdate,'yyyy')) year
7 from dual

```

HOUR	MIN	SEC	DAY	MTH	YEAR
20	28	36	15	6	2005

PostgreSQL

Для извлечения единиц времени из даты используем функции TO_CHAR и TO_NUMBER:

```

1 select to_number(to_char(current_timestamp,'hh24'),'99') as hr,
2        to_number(to_char(current_timestamp,'mi'),'99') as min,
3        to_number(to_char(current_timestamp,'ss'),'99') as sec,

```



```

4     to_number(to_char(current_timestamp,'dd'),'99') as day,
5     to_number(to_char(current_timestamp,'mm'),'99') as mth,
6     to_number(to_char(current_timestamp,'yyyy'),'9999') as yr
7     from t1

```

HR	MIN	SEC	DAY	MTH	YR
20	28	36	15	6	2005

MySQL

Для извлечения единиц времени из даты используем функцию DATE_FORMAT:

```

1 select date_format(current_timestamp,'%k') hr,
2     date_format(current_timestamp,'%i') min,
3     date_format(current_timestamp,'%s') sec,
4     date_format(current_timestamp,'%d') dy,
5     date_format(current_timestamp,'%m') mon,
6     date_format(current_timestamp,'%Y') yr
7     from t1

```

HR	MIN	SEC	DAY	MTH	YR
20	28	36	15	6	2005

SQL Server

Для извлечения единиц времени из даты используем функцию DATEPART::функция:

```

1 select datepart( hour, getdate()) hr,
2     datepart( minute,getdate()) min,
3     datepart( second,getdate()) sec,
4     datepart( day, getdate()) dy,
5     datepart( month, getdate()) mon,
6     datepart( year, getdate()) yr
7     from t1

```

HR	MIN	SEC	DAY	MTH	YR
20	28	36	15	6	2005

Обсуждение

Эти решения ничем особо не отличаются — мы просто используем встроенные средства СУБД. При этом весьма полезно уделить некоторое время ознакомлению со всеми доступными функциями для работы с датами. Рассмотренные в этом рецепте возможности функций являются только вершиной айсберга всех их возможностей. В частности, каждая из этих функций может принимать намного больше аргументов и возвращать больший объем информации, чем было показано в рецепте.

9.4. Вычисление первого и последнего дней месяца

ЗАДАЧА

Требуется вычислить первый и последний дни текущего месяца.

РЕШЕНИЕ

Хотя в приводимых далее решениях рассматривается вычисление первого и последнего дней текущего месяца, выполнив небольшую корректировку, вы сможете использовать их для любого месяца.

DB2

С помощью функции `DAY` определяем количество дней, истекших в текущем месяце по текущую дату. Затем вычитаем полученное значение из текущей даты и добавляем 1, получая первое число текущего месяца. Чтобы получить последний день месяца, добавляем один месяц к текущей дате, а затем вычитаем из полученной величины значение, возвращенное функцией `DAY` для текущей даты:

```
1 select (date(current_date) - day(date(current_date)) day + 1 day) firstday,
2        (date(current_date)+1 month
3         - day(date(current_date)+1 month) day) lastday
4   from t1
```

Oracle

С помощью функции `TRUNC` вычисляем первое число месяца, а с помощью функции `LAST_DAY` — последний:

```
1 select trunc(sysdate,'mm') firstday,
2        last_day(sysdate) lastday
3   from dual
```



При использовании функции `TRUNC` показанным в этом примере способом теряются компоненты времени дня, тогда как функция `LAST_DAY` этого не допускает.

PostgreSQL

Вызвав функцию `DATE_TRUNC`, усекаем текущую дату до первого дня текущего месяца. К полученному значению добавляем один месяц и вычитаем день, чтобы получить последний день текущего месяца:

```
1 select firstday,
2        cast(firstday + interval '1 month'
3              - interval '1 day' as date) as lastday
4   from (
5 select cast(date_trunc('month',current_date) as date) as firstday
```

```
6 from t1
7      ) x
```

MySQL

С помощью функций `DATE_ADD` и `DAY` вычисляем количество дней текущего месяца, прошедших до текущей даты. Затем вычитаем полученное значение из текущей даты и добавляем 1, получая первое число текущего месяца. Наконец, используем функцию `LAST_DAY`, чтобы вычислить последний день месяца:

```
1 select date_add(current_date,
2               interval -day(current_date)+1 day) firstday,
3        last_day(current_date) lastday
4 from t1
```

SQL Server

С помощью функций `DATEADD` и `DAY` вычисляем количество дней текущего месяца, прошедших до текущей даты. Затем вычитаем полученное значение из текущей даты и добавляем 1, получая первое число текущего месяца. Чтобы получить последний день месяца, добавляем один месяц к текущей дате, а затем вычитаем из полученной величины значение, возвращенное функцией `DAY` для текущей даты, снова используя функции `DAY` и `DATEADD`:

```
1 select dateadd(day, -day(getdate()+1, getdate()) firstday, .
2        dateadd(day,
3                -day(dateadd(month, 1, getdate())) ,
4                dateadd(month, 1, getdate())) lastday
5 from t1
```

Обсуждение

DB2

Чтобы получить первый день месяца, просто определяем числовое значение текущего дня месяца, а затем вычитаем это значение из текущей даты. Например, числовое значения для даты 14 марта равно 14. Вычитая 14 дней из 14 марта, получим последний день февраля. Далее просто добавляем один день, чтобы получить первое число текущего месяца. Последний день месяца вычисляется подобно первому: вычитаем числовое значение дня из текущей даты, чтобы получить последний день предыдущего месяца. А чтобы получить последний день текущего месяца, добавляем к этому значению один месяц.

Oracle

Первый день текущего месяца вычисляем с помощью функции `TRUNC`, передавая ей в качестве второго аргумента строку 'mm', чтобы усечь текущий месяц до его первого числа. А последний день текущего месяца находим, просто используя функцию `LAST_DAY`.

PostgreSQL

Первый день текущего месяца вычисляем с помощью функции `DATE_TRUNC`, передавая ей в качестве второго аргумента строку `'month'`, чтобы усесть текущую дату до первого числа месяца. А чтобы вычислить последний день текущего месяца, добавляем один месяц к его первому числу, а затем вычитаем единицу.

MySQL

Первый день текущего месяца находим, используя функцию `DAY`, которая возвращает числовое значение переданной ей даты. Затем вычисляем последний день предыдущего месяца, вычитая из текущей даты значение, возвращенное функцией `DAY(CURRENT_DATE)`. Добавив к полученному значению единицу, получим первый день текущего месяца. А последний день текущего месяца находим, просто используя функцию `LAST_DAY`.

SQL Server

Первый день текущего месяца находим, используя функцию `DAY`, которая возвращает числовое значение переданной ей даты. Затем вычисляем последний день предыдущего месяца, вычитая из текущей даты значение, возвращенное функцией `DAY(GETDATE())`. Добавив к полученному значению единицу, получим первый день текущего месяца. Последний день текущего месяца вычисляем с помощью функции `DATEADD`. В частности, добавляем один месяц к текущей дате, а затем вычитаем из полученной величины значение, возвращенное функцией `DAY(DATEADD(MONTH, 1, GETDATE()))`, получая последний день месяца.

9.5. Вычисление дат определенного дня недели для всего года

ЗАДАЧА

Требуется вычислить даты определенного дня недели для всего года. Например, надо создать список дат для всех пятниц текущего года.

РЕШЕНИЕ

Независимо от используемой СУБД, для решения этой задачи применяется один общий алгоритм: возвращаем все дни текущего года, из которых оставляем только те, которые соответствуют требуемому дню недели. В примерах этого решения таким днем является пятница.

DB2

Для получения всех дней текущего года используем рекурсивный оператор `WITH`, а затем с помощью функции `DAYNAME` оставляем только пятницы:

```

1   with x (dy,yr)
2     as (
3   select dy, year(dy) yr
4     from (
5   select (current_date -
6         dayofyear(current_date) days +1 days) as dy
7     from t1
8        ) tmp1
9   union all
10  select dy+1 days, yr
11     from x
12    where year(dy +1 day) = yr
13 )
14 select dy
15     from x
16    where dayname(dy) = 'Friday'

```

Oracle

Для получения всех дней текущего года используем рекурсивное выражение `CONNECT BY`, а затем с помощью функции `TO_CHAR` оставляем только пятницы:

```

1   with x
2     as (
3   select trunc(sysdate,'y')+level-1 dy
4     from t1
5     connect by level <=
6        add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
7 )
8 select *
9     from x
10    where to_char(dy, 'dy') = 'fri'

```

PostgreSQL

Сначала с помощью обобщенного табличного выражения создаем список всех дней года, а затем отфильтровываем дни, не являющиеся пятницей. В этом решении используется функция `EXTRACT`, отвечающая требованиям стандарта ANSI, поэтому оно может исполняться на широком круге СУБД:

```

1 with recursive cal (dy)
2 as (
3 select current_date
4   -(cast
5     (extract(doy from current_date) as integer)
6     -1)
7 union all
8 select dy+1
9 from cal

```

```

10 where extract(year from dy)=extract(year from (dy+1))
11 )
12
13 select dy,extract(dow from dy) from cal
14 where cast(extract(dow from dy) as integer) = 6

```

MySQL

Сначала с помощью обобщенного табличного выражения создаем список всех дней года, а затем отфильтровываем все дни, не являющиеся пятницей:

```

1      with recursive cal (dy,yr)
2  as
3  (
4    select dy, extract(year from dy) as yr
5  from
6    (select adddate
7      (adddate(current_date, interval - dayofyear(current_date)
8      day), interval 1 day) as dy) as tmp1
9  union all
10   select date_add(dy, interval 1 day), yr
11 from cal
12 where extract(year from date_add(dy, interval 1 day)) = yr
13 )
14   select dy from cal
15   where dayofweek(dy) = 6

```

SQL Server

С помощью рекурсивного оператора WITH получаем все дни текущего года, а затем с помощью функции DAYNAME оставляем только пятницы:

```

1  with x (dy,yr)
2  as (
3  select dy, year(dy) yr
4    from (
5  select getdate()-datepart(dy,getdate()+1 dy
6    from t1
7      ) tmp1
8  union all
9  select dateadd(dd,1,dy), yr
10   from x
11   where year(dateadd(dd,1,dy)) = yr
12 )
13 select x.dy
14   from x
15   where datename(dw,x.dy) = 'Friday'
16 option (maxrecursion 400)

```

Обсуждение

DB2

Чтобы выбрать все пятницы текущего года, сначала нужно получить все дни текущего года. На первом шаге с помощью функции `DAYOFYEAR` определяем первый день текущего года. Для этого вычитаем значение, возвращенное функцией `DAYOFYEAR(CURRENT_DATE)`, из текущей даты, чтобы получить последний день предыдущего года (31 декабря). Затем добавляем единицу, получая первый день текущего года:

```
select (current_date
        dayofyear(current_date) days +1 days) as dy
       from t1
```

```
DY
-----
01-JAN-2005
```

Теперь с помощью оператора `WITH` инкрементируем полученное значение по одному дню, пока не достигнем первого дня следующего года. Результирующее множество будет состоять из всех дней текущего года. Далее приводится соответствующий рекурсивный запрос `X` и часть возвращаемых им строк:

```
with x (dy,yr)
     as (
select dy, year(dy) yr
     from (
select (current_date
        dayofyear(current_date) days +1 days) as dy
     from t1
     ) tmp1
union all
select dy+1 days, yr
     from x
     where year(dy +1 day) = yr
)
select dy
     from x
```

```
DY
-----
01-JAN-2020
...
15-FEB-2020
...
22-NOV-2020
...
31-DEC-2020
```

На последнем шаге используем функцию `DAYNAME`, чтобы оставить только строки, соответствующие пятницам.

Oracle

Чтобы выбрать все пятницы текущего года, сначала нужно получить все дни текущего года. На первом шаге определяем начальную дату года, используя для этого функцию `TRUNC`:

```
select trunc(sysdate, 'y') dy
       from t1
```

```
DY
-----
01-JAN-2020
```

Затем с помощью оператора `CONNECT BY` возвращаем все дни текущего года (создание списка строк посредством оператора `CONNECT BY` подробно рассматривается в *рецепте 10.5*).



В этом рецепте используется оператор `WITH`, но можно также применить и вложенный запрос.

Далее приводится соответствующий рекурсивный запрос `X` и часть возвращаемых им строк:

```
with x
     as (
select trunc(sysdate, 'y')+level-1 dy
       from t1
     connect by level <=
               add_months(trunc(sysdate, 'y'), 12)-trunc(sysdate, 'y')
     )
select *
       from x
```

```
DY
-----
01-JAN-2020
...
15-FEB-2020
...
22-NOV-2020
...
31-DEC-2020
```

Наконец, на последнем шаге применяем функцию `TO_CHAR`, чтобы выбрать только пятницы.

PostgreSQL

Чтобы выбрать все пятницы года, нужно сначала сформировать список всех дней года. Для этого сначала вычисляем первый день года, а затем с помощью рекурсивного обобщенного табличного выражения создаем остальные дни года. Вспомним, что в PostgreSQL требуется использовать ключевое слово `RECURSIVE` для идентификации рекурсивных обобщенных табличных выражений.

На последнем шаге применяем функцию `TO_CHAR`, чтобы выбрать только пятницы.

MySQL

Чтобы выбрать все пятницы текущего года, сначала нужно получить все дни текущего года. На первом шаге определяем первый день текущего года. Для этого вычитаем из текущей даты значение, возвращенное функцией `DAYOFYEAR(CURRENT_DATE)`, чтобы получить последний день предыдущего года. Затем добавляем единицу, получая первый день текущего года:

```
select adddate(
    adddate(current_date,
            interval -dayofyear(current_date) day) ,
    interval 1 day ) dy
from t1
```

```
DY
-----
01-JAN-2020
```

Получив первый день года, используем рекурсивное обобщенное табличное выражение, чтобы создать таблицу всех дней года:

```
with cal (dy) as
(select current
union all
select dy+1
```

```
DY
-----
01-JAN-2020
...
15-FEB-2020
...
22-NOV-2020
...
31-DEC-2020
```

На последнем шаге используем функцию `DAYNAME`, чтобы оставить только строки, соответствующие пятницам.

SQL Server

Чтобы выбрать все пятницы текущего года, сначала нужно получить все дни текущего года. На первом шаге с помощью функции `DATEPART` определяем первый день

текущего года. Для этого вычитаем из текущей даты значение, возвращенное функцией `DATEPART(DY,GETDATE())`, чтобы получить последний день предыдущего года. Затем добавляем единицу, получая первый день текущего года:

```
select getdate()-datepart(dy,getdate()+1 dy
      from t1
```

```
DY
-----
01-JAN-2005
```

С помощью оператора `WITH` и функции `DATEADD` инкрементируем полученное значение по одному дню, пока не достигнем первого дня следующего года. Результирующее множество будет содержать все дни текущего года. Далее приводится соответствующий рекурсивный запрос `X` и часть возвращаемых им строк:

```
with x (dy,yr)
      as (
select dy, year(dy) yr
      from (
select getdate()-datepart(dy,getdate()+1 dy
      from t1
      ) tmp1
union all
select dateadd(dd,1,dy) , yr
      from x
      where year(dateadd(dd,1,dy)) = yr
)
select x.dy
      from x
option (maxrecursion 400)
```

```
DY
-----
01-JAN-2020
```

Наконец, с помощью функции `DATENAME` выбираем только строки, соответствующие пятницам. Для работы этого решения значение переменной `MAXRECURSION` должно быть минимум 366 (фильтр в рекурсивном запросе `X`, ограничивающий количество дней в году, обеспечивает невозможность создания более чем 366 строк).

9.6. Вычисление дат первого и последнего вхождения заданного дня недели в месяце

ЗАДАЧА

Требуется вычислить, например, первый и последний понедельники текущего месяца.

РЕШЕНИЕ

Понедельник и текущий месяц выбраны произвольно. Решения этого рецепта можно использовать с любым днем недели и любым месяцем. Поскольку каждый отдельный день недели повторяется с интервалом в семь дней, определив первое вхождение требуемого дня, второе определяется добавлением 7 дней, а третье — 14 дней. Подобным образом, при наличии последнего вхождения дня недели в месяце третье вхождение можно вычислить, отняв 7 дней, а второе — 14 дней.

DB2

С помощью рекурсивного оператора WITH генерируем список всех дней текущего месяца, а затем с помощью оператора CASE выделяем все понедельники. Первый и последний понедельники будут самой ранней и самой поздней из выделенных дат:

```

1   with x (dy,mth,is_monday)
2     as (
3 select dy,month(dy),
4         case when dayname(dy)='Monday'
5             then 1 else 0
6         end
7   from (
8 select (current_date-day(current_date) day +1 day) dy
9   from t1
10        ) tmp1
11  union all
12 select (dy +1 day), mth,
13         case when dayname(dy +1 day)='Monday'
14             then 1 else 0
15         end
16   from x
17  where month(dy +1 day) = mth
18 )
19 select min(dy) first_monday, max(dy) last_monday
20   from x
21  where is_monday = 1

```

Oracle

Чтобы вычислить первый и последний понедельники месяца, используем функции NEXT_DAY и LAST_DAY в сочетании с некоторыми хитроумными операциями над датами:

```

select next_day(trunc(sysdate,'mm')-1,'MONDAY') first_monday,
       next_day(last_day(trunc(sysdate,'mm'))-7,'MONDAY') last_monday
from dual

```

PostgreSQL

Сначала определяем первый день текущего месяца, используя для этого функцию `DATE_TRUNC`. Выполнив над полученным значением простые арифметические операции с использованием числовых значений дней недели (дням с воскресенья по понедельник соответствуют числовые значения с 1 по 7), вычисляем первый и последний понедельники текущего месяца:

```

1 select first_monday,
2     case to_char(first_monday+28,'mm')
3         when mth then first_monday+28
4             else first_monday+21
5     end as last_monday
6 from (
7 select case sign(cast(to_char(dy,'d') as integer)-2)
8         when 0
9         then dy
10        when -1
11        then dy+abs(cast(to_char(dy,'d') as integer)-2)
12        when 1
13        then (7-(cast(to_char(dy,'d') as integer)-2))+dy
14        end as first_monday,
15        mth
16 from (
17 select cast(date_trunc('month',current_date) as date) as dy,
18        to_char(current_date,'mm') as mth
19 from t1
20     ) x
21     ) y

```

MySQL

Вычисляем первый день текущего месяца, используя для этого функцию `ADDDATE`. Выполнив над полученным значением простые арифметические операции с использованием числовых значений дней недели (дням с воскресенья по понедельник соответствуют числовые значения с 1 по 7), вычисляем первый и последний понедельники текущего месяца:

```

1 select first_monday,
2     case month(adddate(first_monday,28))
3         when mth then adddate(first_monday,28)
4             else adddate(first_monday,21)
5     end last_monday
6 from (
7 select case sign(dayofweek(dy)-2)
8         when 0 then dy
9         when -1 then adddate(dy,abs(dayofweek(dy)-2))

```

```

10         when 1 then adddate(dy, (7-(dayofweek(dy)-2)))
11     end first_monday,
12     mth
13 from (
14 select adddate(adddate(current_date,-day(current_date)),1) dy,
15         month(current_date) mth
16 from t1
17     ) x
18     ) y

```

SQL Server

С помощью рекурсивного оператора WITH генерируем список всех дней текущего месяца, а затем с помощью оператора CASE выделяем все понедельники. Первый и последний понедельники будут самой ранней и самой поздней из выделенных дат:

```

1  with x (dy,mth,is_monday)
2  as (
3  select dy,mth,
4         case when datepart(dw,dy) = 2
5             then 1 else 0
6         end
7  from (
8  select dateadd(day,1,dateadd(day,-day(getdate()),getdate())) dy,
9         month(getdate()) mth
10 from t1
11     ) tmp1
12 union all
13 select dateadd(day,1,dy),
14         mth,
15         case when datepart(dw,dateadd(day,1,dy)) = 2
16             then 1 else 0
17         end
18 from x
19 where month(dateadd(day,1,dy)) = mth
20 )
21 select min(dy) first_monday,
22         max(dy) last_monday
23 from x
24 where is_monday = 1

```

Обсуждение

DB2 и SQL Server

Хотя для решения этой задачи в DB2 и SQL Server используются разные функции, сам метод решения абсолютно одинаков. Если внимательно рассмотреть оба решения, то можно увидеть, что единственная разница между ними состоит в том, как

выполняется сложение дат. Это обсуждение охватывает оба решения, но использует код решения для DB2 для отображения результатов промежуточных шагов.



Если используемая вами версия SQL Server или DB2 не поддерживает рекурсивный оператор `WITH`, вместо него можно использовать метод решения для PostgreSQL.

Первый шаг в вычислении первого и последнего понедельников текущего месяца — получить первый день месяца. Это осуществляется вложенным запросом `tmp1` в рекурсивном представлении `X`, где сначала определяется текущая дата, или, более точно, день месяца для текущей даты. День месяца для текущей даты представляет количество дней, истекших с начала месяца. Например, для 10 апреля это будет 10 дней. Вычитая это значение из текущей даты, получаем последний день предыдущего месяца. Например, вычитая 10 из 10 апреля, получаем последний день марта. Теперь, чтобы получить первый день текущего месяца, просто добавляем единицу к полученному результату:

```
select (current_date-day(current_date) day +1 day) dy
       from t1
```

```
DY
-----
01-JUN-2005
```

Затем с помощью функции `MONTH` определяем месяц текущей даты, а посредством простого выражения `CASE` определяем, является ли первый день месяца понедельником:

```
select dy, month(dy) mth,
       case when dayname(dy)='Monday'
            then 1 else 0
       end is_monday
       from (
select (current_date-day(current_date) day +1 day) dy
       from t1
       ) tmp1
```

```
DY          MTH  IS_MONDAY
-----
01-JUN-2005 6          0
```

Теперь с помощью рекурсивного оператора `WITH` инкрементируем значение первого дня месяца по одному дню, пока не достигнем первого дня следующего месяца. При этом с помощью оператора `CASE` определяем, какие дни месяца являются понедельником, помечая их флагом 1. Далее приводится соответствующий рекурсивный запрос `X` и часть возвращаемых им строк:

```
with x (dy,mth,is_monday)
      as (
select dy,month(dy) mth,
```

```

        case when dayname(dy)='Monday'
            then 1 else 0
        end is_monday
    from (
select (current_date-day(current_date) day +1 day) dy
    from t1
        ) tmp1
    union all
select (dy +1 day), mth,
        case when dayname(dy +1 day)='Monday'
            then 1 else 0
        end
    from x
    where month(dy +1 day) = mth
)
select *
    from x

```

DY	MTH	IS_MONDAY
-----	-----	-----
01-JUN-2005	6	0
02-JUN-2005	6	0
03-JUN-2005	6	0
04-JUN-2005	6	0
05-JUN-2005	6	0
06-JUN-2005	6	1
07-JUN-2005	6	0
08-JUN-2005	6	0

Поскольку значение столбца IS_MONDAY равно 1 только для понедельников, то в завершение применяем агрегатные функции MIN и MAX к строкам, в которых значение IS_MONDAY равно 1, чтобы определить первый и последний понедельники месяца.

Oracle

Наличие в Oracle функции NEXT_DAY значительно облегчает решение этой задачи. Чтобы найти первый понедельник текущего месяца, сначала возвращаем последний день предыдущего месяца посредством арифметических операций с использованием функции TRUNC:

```

select trunc(sysdate,'mm')-1 dy
    from dual

```

```

DY
-----
31-MAY-2005

```

Затем с помощью функции NEXT_DAY находим первый понедельник после последнего дня предыдущего месяца (т. е. первый понедельник текущего месяца):

```
select next_day(trunc(sysdate,'mm')-1,'MONDAY') first_monday
      from dual
```

```
FIRST_MONDAY
-----
06-JUN-2005
```

Чтобы найти последний понедельник текущего месяца, сначала с помощью функции `TRUNC` возвращаем первый день текущего месяца:

```
select trunc(sysdate,'mm') dy
      from dual
```

```
DY
-----
01-JUN-2005
```

А на следующем шаге вычисляем последнюю неделю (последние семь дней) текущего месяца. Для этого посредством функции `LAST_DAY` находим последний день месяца, а затем вычитаем из него семь дней:

```
select last_day(trunc(sysdate,'mm'))-7 dy
      from dual
```

```
DY
-----
23-JUN-2005
```

Если сразу не очевидно, что здесь происходит, возвращаемся на семь дней назад от последнего дня текущего месяца, чтобы обеспечить наличие в месяце хотя бы одного какого-либо дня недели. На последнем шаге с помощью функции `NEXT_DAY` находим следующий (и последний) понедельник месяца:

```
select next_day(last_day(trunc(sysdate,'mm'))-7,'MONDAY') last_monday
      from dual
```

```
LAST_MONDAY
-----
27-JUN-2005
```

PostgreSQL и MySQL

Решения для PostgreSQL и MySQL также основаны на одинаковом подходе и различаются только применяемыми функциями. Соответствующие запросы решений чрезвычайно просты, несмотря на их значительный размер. Однако нахождение первого и последнего понедельников текущего месяца связано с некоторыми накладными издержками.

На первом шаге определяем первый день текущего месяца, после чего нужно найти первый понедельник месяца. Поскольку эти СУБД не поддерживают функцию для нахождения следующей даты для заданного дня недели, приходится выполнить несколько арифметических операций. Выражение `CASE`, начинающееся на строке 7 в обоих решениях, вычисляет разницу между цифровым значением дня недели пер-

вого дня месяца и числовым значением, соответствующим понедельнику. При вызове с указанием формата 'd' (или 'D') функция `TO_CHAR` для PostgreSQL возвращает числовые значения в диапазоне от 1 до 7, представляющие дни недели от воскресенья до субботы. Функция `DAYOFWEEK` для MySQL возвращает такие же результаты. Таким образом, понедельнику всегда соответствует значение 2. Выражение `CASE` сначала проверяет результат `SIGN` вычитания из числового значения первого дня месяца (каким бы оно ни было) числового значения понедельника (равное 2). Если результат 0, тогда первый день месяца является понедельником, который и будет первым понедельником месяца. Если результат -1 , тогда первым днем месяца является воскресенье, и чтобы определить первый понедельник месяца, просто добавляем к первому дню месяца разницу в днях между 2 и 1 (числовые значения понедельника и воскресенья, соответственно).



Если вам трудно понять, что здесь происходит, игнорируйте названия дней недели и просто выполняйте арифметические операции. Предположим, что первым днем месяца является вторник и нам нужно найти следующую пятницу. Функция `TO_CHAR` (PostgreSQL) с указанием формата 'd' и функция `DAYOFWEEK` (MySQL) возвращают для пятницы значение 6, а для вторника — 3. Чтобы из 3 получить 6, просто вычитаем меньшее значение из большего ($6 - 3 = 3$) и добавляем полученный результат к меньшему значению ($(6 - 3) + 3 = 6$). Таким образом, независимо от реальных дат, если числовое значение начального дня недели меньше, чем числовое значения искомого дня недели, получить искомую дату можно, сложив разницу между этими двумя днями с начальной датой.

Если результат `SIGN` равен 1, тогда первый день месяца находится между вторником и субботой, включая эти дни. Если числовое значение первого дня месяца больше чем 2 (понедельник), вычитаем из значения 7 разницу между числовым значением первого дня месяца и числовым значением понедельника (2) и суммируем это значение с первым днем месяца. Таким образом, мы получили требуемый день недели, в нашем случае — понедельник.



Опять же, если вам трудно понять, что происходит здесь, игнорируйте названия дней недели и просто выполняйте арифметические операции. Предположим, что нужно найти следующий вторник, начиная с пятницы. Числовое значение вторника (3) меньше, чем числовое значение пятницы (6). Чтобы из 6 получить 3, вычитаем из 7 разницу между этими двумя значениями ($7 - (6 - 3) = 4$) и суммируем полученный результат с начальным днем (пятницей). Вертикальные черточки в выражении $|3 - 6|$ обозначают абсолютное значение этой операции вычитания. В нашем случае мы не суммируем 4 и 6, что дало бы 10. Мы просто добавляем четыре дня к пятнице, что даст нам следующий вторник.

Идея, положенная в основу выражения `CASE`, заключается в создании своего рода функции `NEXTDAY` (следующий день) для PostgreSQL и MySQL. Если начинать не с первого дня месяца, `BY` будет присваиваться значение, возвращаемое функцией `CURRENT_DAY`, а результатом выражения `CASE` будет дата понедельника, следующего после текущего дня. (Но если `CURRENT_DAY` возвратит понедельник, тогда результатом `CASE` будет дата этого понедельника.)

Получив первый понедельник месяца, добавляем к нему 21 или 28 дней, чтобы найти последний понедельник. Сколько именно дней нужно добавлять — 21 или 28, —

определяется выражением `CASE` в строках 2–5, которое проверяет, вызывает ли добавление 28 переход в следующий месяц. Выражение `CASE` осуществляет эту проверку следующим образом:

1. Добавляет 28 к значению `FIRST_MONDAY`.
2. Из полученной суммы `FIRST_MONDAY+28` извлекается название текущего месяца с помощью функции `TO_CHAR` (PostgreSQL) или `MONTH` (MySQL).
3. Полученный результат сравнивается со значением `MTH` из вложенного представления. Значение `MTH` — это название текущего месяца, возвращенное функцией `CURRENT_DATE`. Совпадение двух значений двух месяцев означает, что месяц содержит достаточное количество дней, чтобы к нему можно было добавить 28 дней, не выходя за его пределы, и выражение `CASE` возвращает `FIRST_MONDAY+28`. А несовпадение этих двух значений означает, что при добавлении к нему 28 дней мы выйдем за пределы месяца, и выражение `CASE` возвращает `FIRST_MONDAY+21`. В рассматриваемом случае очень удобно, что наши месяцы обладают таким свойством, и нам нужно проверять лишь два возможных значения: 28 и 21.



Решение можно расширить, добавляя 7 и 14 дней для нахождения второго и третьего понедельников, соответственно.

9.7. Создание календаря

ЗАДАЧА

Требуется создать календарь для текущего месяца. Оформление календаря должно быть аналогичным обычному календарю: семь столбцов дней недели и пять (обычно) строк.

РЕШЕНИЕ

Каждое решение будет выглядеть немного иначе, но во всех применяется одинаковый подход: возвращаются все дни текущего месяца, после чего все дни каждой недели транспонируются из строк в столбцы, создавая календарь.

Существуют разные форматы календарей. Например, UNIX-команда `cal` формирует неделю, начинающуюся с воскресенья и заканчивающуюся в субботу. Примеры в этом рецепте основаны на стандарте ISO, поэтому в них неделя начинается с понедельника. Однако, разобравшись с работой решений, вы увидите, что календарь можно сформировать любым требуемым образом, просто исправляя значения, соответствующие стандарту ISO, перед тем, как транспонировать строки в столбцы.



Скорее всего, вы заметите, что использование разных типов SQL-форматирования для создания удобочитаемых результатов увеличивает размер соответствующих запросов. Не стоит пугаться таких запросов, поскольку они, по сути, состоят из нескольких чрезвычайно простых подзапросов.

DB2

Посредством рекурсивного оператора `WITH` получаем все дни текущего месяца, а затем транспонируем полученные строки в столбцы по дням недели, используя выражение `CASE` и функцию `MAX`:

```

1   with x(dy, dm, mth, dw, wk)
2   as (
3   select (current_date -day(current_date) day +1 day) dy,
4          day((current_date -day(current_date) day +1 day)) dm,
5          month(current_date) mth,
6          dayofweek(current_date -day(current_date) day +1 day) dw,
7          week_iso(current_date -day(current_date) day +1 day) wk
8   from t1
9   union all
10  select dy+1 day, day(dy+1 day), mth,
11         dayofweek(dy+1 day), week_iso(dy+1 day)
12   from x
13  where month(dy+1 day) = mth
14 )
15 select max(case dw when 2 then dm end) as Mo,
16        max(case dw when 3 then dm end) as Tu,
17        max(case dw when 4 then dm end) as We,
18        max(case dw when 5 then dm end) as Th,
19        max(case dw when 6 then dm end) as Fr,
20        max(case dw when 7 then dm end) as Sa,
21        max(case dw when 1 then dm end) as Su
22   from x
23  group by wk
24  order by wk

```

Oracle

С помощью рекурсивного выражения `CONNECT BY` получаем все дни текущего месяца, а затем транспонируем полученные строки в столбцы по дням недели, используя выражение `CASE` и функцию `MAX`:

```

1   with x
2   as (
3   select *
4   from (
5   select to_char(trunc(sysdate, 'mm')+level-1, 'iw') wk,
6          to_char(trunc(sysdate, 'mm')+level-1, 'dd') dm,
7          to_number(to_char(trunc(sysdate, 'mm')+level-1, 'd')) dw,
8          to_char(trunc(sysdate, 'mm')+level-1, 'mm') curr_mth,
9          to_char(sysdate, 'mm') mth
10  from dual
11 connect by level <= 31
12 )

```

```

13 where curr_mth = mth
14 )
15 select max(case dw when 2 then dm end) Mo,
16         max(case dw when 3 then dm end) Tu,
17         max(case dw when 4 then dm end) We,
18         max(case dw when 5 then dm end) Th,
19         max(case dw when 6 then dm end) Fr,
20         max(case dw when 7 then dm end) Sa,
21         max(case dw when 1 then dm end) Su
22     from x
23  group by wk
24  order by wk

```

PostgreSQL

С помощью функции `GENERATE_SERIES` получаем все дни текущего месяца, а затем транспонируем полученные строки в столбцы по дням недели, используя выражение `CASE` и функцию `MAX`:

```

1 select max(case dw when 2 then dm end) as Mo,
2         max(case dw when 3 then dm end) as Tu,
3         max(case dw when 4 then dm end) as We,
4         max(case dw when 5 then dm end) as Th,
5         max(case dw when 6 then dm end) as Fr,
6         max(case dw when 7 then dm end) as Sa,
7         max(case dw when 1 then dm end) as Su
8     from (
9  select *
10     from (
11  select cast(date_trunc('month',current_date) as date)+x.id,
12         to_char(
13             cast(
14                 date_trunc('month',current_date)
15                 as date)+x.id,'iw') as wk,
16         to_char(
17             cast(
18                 date_trunc('month',current_date)
19                 as date)+x.id,'dd') as dm,
20         cast(
21             to_char(
22                 cast(
23                     date_trunc('month',current_date)
24                     as date)+x.id,'d') as integer) as dw,
25         to_char(
26             cast(
27                 date_trunc('month',current_date)
28                 as date)+x.id,'mm') as curr_mth,
29         to_char(current_date,'mm') as mth

```

```

30     from generate_series (0,31) x(id)
31         ) x
32     where mth = curr_mth
33         ) y
34     group by wk
35     order by wk

```

MySQL

С помощью рекурсивного обобщенного выражения получаем все дни текущего месяца, а затем транспонируем полученные строки в столбцы по дням недели, используя выражение **CASE** и функцию **MAX**:

```

with recursive x(dy, dm, mth, dw, wk)
    as (
    select dy,
           day(dy) dm,
           datepart(m, dy) mth,
           datepart(dw, dy) dw,
           case when datepart(dw, dy) = 1
                then datepart(wk, dy) - 1
                else datepart(wk, dy)
           end wk
    from (
    select date_add(day, -day(getdate()) + 1, getdate()) dy
    from t1
    ) x
    union all
select dateadd(d, 1, dy), day(date_add(d, 1, dy)), mth,
       datepart(dw, dateadd(d, 1, dy)),
       case when datepart(dw, date_add(d, 1, dy)) = 1
            then datepart(wk, date_add(d, 1, dy)) - 1
            else datepart(wk, date_add(d, 1, dy))
       end
    from x
    where datepart(m, date_add(d, 1, dy)) = mth
)
select max(case dw when 2 then dm end) as Mo,
       max(case dw when 3 then dm end) as Tu,
       max(case dw when 4 then dm end) as We,
       max(case dw when 5 then dm end) as Th,
       max(case dw when 6 then dm end) as Fr,
       max(case dw when 7 then dm end) as Sa,
       max(case dw when 1 then dm end) as Su
    from x
group by wk
order by wk;

```

SQL Server

С помощью рекурсивного оператора WITH получаем все дни текущего месяца, а затем транспонируем полученные строки в столбцы по дням недели, используя выражение CASE и функцию MAX:

```

1  with x(dy, dm, mth, dw, wk)
2    as (
3  select dy,
4         day(dy) dm,
5         datepart(m, dy) mth,
6         datepart(dw, dy) dw,
7         case when datepart(dw, dy) = 1
8              then datepart(wk, dy) - 1
9              else datepart(wk, dy)
10        end wk
11    from (
12  select dateadd(day, -day(getdate()) + 1, getdate()) dy
13    from t1
14    ) x
15  union all
16  select dateadd(d, 1, dy), day(dateadd(d, 1, dy)), mth,
17         datepart(dw, dateadd(d, 1, dy)),
18         case when datepart(dw, dateadd(d, 1, dy)) = 1
19              then datepart(wk, dateadd(d, 1, dy)) - 1
20              else datepart(wk, dateadd(d, 1, dy))
21        end
22    from x
23  where datepart(m, dateadd(d, 1, dy)) = mth
24 )
25  select max(case dw when 2 then dm end) as Mo,
26         max(case dw when 3 then dm end) as Tu,
27         max(case dw when 4 then dm end) as We,
28         max(case dw when 5 then dm end) as Th,
29         max(case dw when 6 then dm end) as Fr,
30         max(case dw when 7 then dm end) as Sa,
31         max(case dw when 1 then dm end) as Su
32    from x
33  group by wk
34  order by wk

```

Обсуждение

DB2

На первом шаге нужно вернуть все дни месяца, для которого создается календарь. Эта задача решается с помощью рекурсивного оператора WITH. Вместе с каждым днем месяца (DM) также нужно вернуть все составляющие его даты: день

недели (DW), текущий месяц (MTH) и ISO-номер недели для каждого дня месяца (WK). Далее приводятся результаты, возвращаемые рекурсивным представлением X до проведения рекурсии (верхняя часть оператора UNION ALL):

```
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
from t1
```

DY	DM	MTH	DW	WK
01-JUN-2005	01	06	4	22

На следующем шаге инкрементируем значение для DM (перебираем дни месяца), пока не достигнем первого дня следующего месяца. При этом для каждого дня также возвращается его день недели и ISO-номер его недели. Далее приводятся соответствующий запрос и его частичные результаты:

```
with x(dy, dm, mth, dw, wk)
as (
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
from t1
union all
select dy+1 day, day(dy+1 day), mth,
       dayofweek(dy+1 day), week_iso(dy+1 day)
from x
where month(dy+1 day) = mth
)
select *
from x
```

DY	DM	MTH	DW	WK
01-JUN-2020	01	06	4	22
02-JUN-2020	02	06	5	22
...				
21-JUN-2020	21	06	3	25
22-JUN-2020	22	06	4	25
...				
30-JUN-2020	30	06	5	26

На этом этапе возвращаются следующие данные: все дни текущего месяца, двухцифровые числовые значения всех дней, двухцифровое числовое значение текущего месяца, одноцифровые числовые значения дней недели (в диапазоне от 1 до 7,

представляющие дни с воскресенья по субботу, соответственно) и двухцифровой ISO-номер недели, в которую входит каждый день. Применяя к этой информации выражение CASE, мы можем определить день недели для каждого значения DM (день месяца). Соответствующий запрос и часть его результатов приведены далее:

```
with x(dy, dm, mth, dw, wk)
  as (
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
  from t1
 union all
select dy+1 day, day(dy+1 day), mth,
       dayofweek(dy+1 day), week_iso(dy+1 day)
  from x
 where month(dy+1 day) = mth
 )
select wk,
       case dw when 2 then dm end as Mo,
       case dw when 3 then dm end as Tu,
       case dw when 4 then dm end as We,
       case dw when 5 then dm end as Th,
       case dw when 6 then dm end as Fr,
       case dw when 7 then dm end as Sa,
       case dw when 1 then dm end as Su
  from x
```

```
WK MO TU WE TH FR SA SU
-- -- -- -- -- -- --
22      01
22      02
22      03
22      04
22      05
23 06
23 07
23      08
23      09
23      10
23      11
23      12
```

Как можно видеть по частичному результату, каждый день каждой недели возвращается в виде отдельной строки. Теперь нам нужно сгруппировать дни по неделям, а затем собрать все дни каждой недели в одну строку. Для транспонирования строк дней недели в одну строку используем агрегатную функцию MAX с группированием

по `WK` (ISO-неделя). Чтобы отформатировать календарь должным образом и обеспечить правильный порядок дней, упорядочиваем результаты по значению `WK`. Соответствующий запрос и его результаты приведены далее:

```
with x(dy, dm, mth, dw, wk)
  as (
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
  from t1
 union all
select dy+1 day, day(dy+1 day), mth,
       dayofweek(dy+1 day), week_iso(dy+1 day)
  from x
  where month(dy+1 day) = mth
 )
select max(case dw when 2 then dm end) as Mo,
       max(case dw when 3 then dm end) as Tu,
       max(case dw when 4 then dm end) as We,
       max(case dw when 5 then dm end) as Th,
       max(case dw when 6 then dm end) as Fr,
       max(case dw when 7 then dm end) as Sa,
       max(case dw when 1 then dm end) as Su
  from x
 group by wk
 order by wk
```

```
MO TU WE TH FR SA SU
-- -- -- -- -- -- --
      01 02 03 04 05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

Oracle

На первом шаге с помощью рекурсивного оператора `CONNECT BY` создаем строки всех дней месяца, для которого создаем календарь. Но оператор `CONNECT BY` не поддерживается версиями Oracle более ранними, чем Oracle9i. Поэтому вместо него можно использовать сводную таблицу — например, T500, как в решении для MySQL.

Вместе с каждым днем месяца также нужно вернуть все числовые значения для составляющих его даты: день месяца (`DM`), день недели (`DW`), текущий месяц (`MTH`) и ISO-номер недели для каждого дня месяца (`WK`). Далее приводятся результаты представления `X` оператора `WITH` для первого дня текущего месяца:

```

select trunc(sysdate,'mm') dy,
       to_char(trunc(sysdate,'mm'),'dd') dm,
       to_char(sysdate,'mm') mth,
       to_number(to_char(trunc(sysdate,'mm'),'d')) dw,
       to_char(trunc(sysdate,'mm'),'iw') wk
from dual

```

```

DY          DM MT          DW WK
-----
01-JUN-2020 01 06          4 22

```

На следующем шаге инкрементируем значение для DM (перебираем дни месяца), пока не достигнем первого дня следующего месяца. При этом также возвращается день недели для каждого дня и ISO-номер его недели. Далее приводится соответствующий запрос и его частичные результаты (для удобочитаемости к каждому дню добавлена его полная дата).

```

with x
  as (
select *
  from (
select trunc(sysdate,'mm')+level-1 dy,
       to_char(trunc(sysdate,'mm')+level-1,'iw') wk,
       to_char(trunc(sysdate,'mm')+level-1,'dd') dm,
       to_number(to_char(trunc(sysdate,'mm')+level-1,'d')) dw,
       to_char(trunc(sysdate,'mm')+level-1,'mm') curr_mth,
       to_char(sysdate,'mm') mth
  from dual
 connect by level <= 31
        )
  where curr_mth = mth
)
select *
  from x

```

```

DY          WK DM          DW CU MT
-----
01-JUN-2020 22 01          4 06 06
02-JUN-2020 22 02          5 06 06
...
21-JUN-2020 25 21          3 06 06
22-JUN-2020 25 22          4 06 06
...
30-JUN-2020 26 30          5 06 06

```

На этом этапе для каждого дня текущего месяца возвращается по одной строке, содержащей следующие данные: двухцифровое числовое значение дня месяца, двухцифровое числовое значение текущего месяца, одноцифровое числовое значение дня недели (в диапазоне от 1 до 7, представляющее дни с воскресенья по субботу,

соответственно) и двухцифровой ISO-номер недели, в которую входит этот день. Применяя к этой информации выражение `CASE`, мы можем определить день недели для каждого значения `DM` (день месяца). Соответствующий запрос и часть его результатов приведены далее:

```
with x
  as (
select *
  from (
select trunc(sysdate,'mm')+level-1 dy,
       to_char(trunc(sysdate,'mm')+level-1,'iw') wk,
       to_char(trunc(sysdate,'mm')+level-1,'dd') dm,
       to_number(to_char(trunc(sysdate,'mm')+level-1,'d')) dw,
       to_char(trunc(sysdate,'mm')+level-1,'mm') curr_mth,
       to_char(sysdate,'mm') mth
  from dual
connect by level <= 31
        )
  where curr_mth = mth
)
select wk,
       case dw when 2 then dm end as Mo,
       case dw when 3 then dm end as Tu,
       case dw when 4 then dm end as We,
       case dw when 5 then dm end as Th,
       case dw when 6 then dm end as Fr,
       case dw when 7 then dm end as Sa,
       case dw when 1 then dm end as Su
  from x
```

```
WK MO TU WE TH FR SA SU
-- -- -- -- -- -- --
22      01
22      02
22      03
22      04
22      05
23 06
23 07
23 08
23 09
23 10
23 11
23 12
```

Как можно видеть, номер каждого дня месяца возвращается в отдельной строке, но также и в отдельном столбце, соответствующем дню недели. Теперь нам нужно свести все дни каждой недели в одну строку. Для транспонирования строк дней

недели в одну строку используем агрегатную функцию `MAX` с группированием по `WK` (ISO-неделя). Чтобы отформатировать календарь должным образом и обеспечить правильный порядок дней, упорядочиваем результаты по значению `WK`. Соответствующий запрос и его конечные результаты приведены далее:

```
with x
  as (
select *
  from (
select to_char(trunc(sysdate,'mm')+level-1,'iw') wk,
       to_char(trunc(sysdate,'mm')+level-1,'dd') dm,
       to_number(to_char(trunc(sysdate,'mm')+level-1,'d')) dw,
       to_char(trunc(sysdate,'mm')+level-1,'mm') curr_mth,
       to_char(sysdate,'mm') mth
  from dual
connect by level <= 31
         )
  where curr_mth = mth
)
select max(case dw when 2 then dm end) Mo,
       max(case dw when 3 then dm end) Tu,
       max(case dw when 4 then dm end) We,
       max(case dw when 5 then dm end) Th,
       max(case dw when 6 then dm end) Fr,
       max(case dw when 7 then dm end) Sa,
       max(case dw when 1 then dm end) Su
  from x
group by wk
order by wk
```

```
MO TU WE TH FR SA SU
-- -- -- -- -- --
      01 02 03 04 05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

MySQL, PostgreSQL и SQL Server

Во всех этих решениях используется одинаковый подход, только для возвращения дат применяются разные функции. Поясним этот подход на примере решения для SQL Server. На первом шаге мы возвращаем по одной строке для каждого дня месяца, используя для этого рекурсивный оператор `WITH`. Для каждой возвращаемой строки дня нам также нужно вернуть все числовые значения для составляющих его даты: день месяца (`DM`), день недели (`DW`), текущий месяц (`MTH`) и ISO-номер недели для каждого дня месяца (`WK`). Далее приводятся результаты, возвращаемые

рекурсивным представлением X до проведения рекурсии (верхняя часть оператора UNION ALL):

```
select dy,
       day(dy) dm,
       datepart(m,dy) mth,
       datepart(dw,dy) dw,
       case when datepart(dw,dy) = 1
            then datepart(wv,dy)-1
            else datepart(wv,dy)
       end wk
from (
select dateadd(day,-day(getdate()+1,getdate()) dy
from t1
) x
```

DY	DM	MTH	DW	WK
01-JUN-2005	1	6	4	23

На следующем шаге инкрементируем значение для DM (перебираем дни месяца), пока не достигнем первого дня следующего месяца. При этом также возвращается день недели для каждого дня и ISO-номер его недели. Далее приводятся соответствующий запрос и его частичные результаты:

```
with x(dy, dm, mth, dw, wk)
as (
select dy,
       day(dy) dm,
       datepart(m,dy) mth,
       datepart(dw,dy) dw,
       case when datepart(dw,dy) = 1
            then datepart(wv,dy)-1
            else datepart(wv,dy)
       end wk
from (
select dateadd(day,-day(getdate()+1,getdate()) dy
from t1
) x
union all
select dateadd(d,1,dy), day(dateadd(d,1,dy)), mth,
       datepart(dw,dateadd(d,1,dy)),
       case when datepart(dw,dateadd(d,1,dy)) = 1
            then datepart(wk,dateadd(d,1,dy))-1
            else datepart(wk,dateadd(d,1,dy))
       end
from x
where datepart(m,dateadd(d,1,dy)) = mth
)
```

```
select *
  from x
```

DY	DM	MTH	DW	WK
01-JUN-2005	01	06	4	23
02-JUN-2005	02	06	5	23
...				
21-JUN-2005	21	06	3	26
22-JUN-2005	22	06	4	26
...				
30-JUN-2005	30	06	5	27

Теперь для каждого дня текущего месяца у нас есть следующие данные: двухцифровое числовое значения дня месяца, двухцифровое числовое значение текущего месяца, одноцифровое числовое значение дня недели (в диапазоне от 1 до 7, представляющее дни с воскресенья по субботу, соответственно) и двухцифровой ISO-номер недели, в которую входит этот день.

Применяя к этой информации выражение `CASE`, мы можем определить день недели для каждого значения `DM` (день месяца). Соответствующий запрос и часть его результатов приведены далее:

```
with x(dy, dm, mth, dw, wk)
  as (
select dy,
       day(dy) dm,
       datepart(m, dy) mth,
       datepart(dw, dy) dw,
       case when datepart(dw, dy) = 1
            then datepart(dw, dy) - 1
            else datepart(dw, dy)
       end wk
  from (
select dateadd(day, -day(getdate()) + 1, getdate()) dy
  from t1
       ) x
  union all
select dateadd(d, 1, dy), day(dateadd(d, 1, dy)), mth,
       datepart(dw, dateadd(d, 1, dy)),
       case when datepart(dw, dateadd(d, 1, dy)) = 1
            then datepart(dw, dateadd(d, 1, dy)) - 1
            else datepart(dw, dateadd(d, 1, dy))
       end
  end
  from x
  where datepart(m, dateadd(d, 1, dy)) = mth
)
select case dw when 2 then dm end as Mo,
       case dw when 3 then dm end as Tu,
       case dw when 4 then dm end as We,
```

```

    case dw when 5 then dm end as Th,
    case dw when 6 then dm end as Fr,
    case dw when 7 then dm end as Sa,
    case dw when 1 then dm end as Su

```

```

from x

```

```

WK MO TU WE TH FR SA SU
-- -- -- -- -- -- --
22      01
22      02
22      03
22      04
22      05
23 06
23  07
23   08
23    09
23   10
23   11
23   12

```

Каждый день месяца возвращается в отдельной строке, где содержащий номер дня столбец соответствует этому дню недели. Теперь нам нужно свести все дни каждой недели в одну строку. Для этого группируем строки по `wk` (ISO-неделя) и применяем агрегатную функцию `max` к разным столбцам. Соответствующий запрос и его конечный результат показаны далее:

```

with x(dy, dm, mth, dw, wk)
  as (
select dy,
       day(dy) dm,
       datepart(m, dy) mth,
       datepart(dw, dy) dw,
       case when datepart(dw, dy) = 1
            then datepart(wk, dy) - 1
            else datepart(wk, dy)
       end wk
  from (
select dateadd(day, -day(getdate()) + 1, getdate()) dy
  from t1
  ) x
  union all
select dateadd(d, 1, dy), day(dateadd(d, 1, dy)), mth,
       datepart(dw, dateadd(d, 1, dy)),
       case when datepart(dw, dateadd(d, 1, dy)) = 1
            then datepart(wk, dateadd(d, 1, dy)) - 1
            else datepart(wk, dateadd(d, 1, dy))
       end
  from x

```

```

where datepart(m,dateadd(d,1,dy)) = mth
)
select max(case dw when 2 then dm end) as Mo,
       max(case dw when 3 then dm end) as Tu,
       max(case dw when 4 then dm end) as We,
       max(case dw when 5 then dm end) as Th,
       max(case dw when 6 then dm end) as Fr,
       max(case dw when 7 then dm end) as Sa,
       max(case dw when 1 then dm end) as Su
  from x
 group by wk
 order by wk

```

```

MO TU WE TH FR SA SU
-- -- -- -- -- -- --
01 02 03 04 05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

```

9.8. Создание списка начальных и конечных дат кварталов года

ЗАДАЧА

Требуется вернуть список начальных и конечных дат для всех четырех кварталов заданного года.

РЕШЕНИЕ

Поскольку в году четыре квартала, нам нужно создать четыре строки. Создав требуемое количество строк, просто используем встроенные функции используемой СУБД, чтобы вернуть начальные и конечные даты соответствующих кварталов. Наша цель — вернуть результирующее множество такого вида (текущий год выбран абсолютно произвольно):

```

QTR Q_START      Q_END
-----
 1 01-JAN-2020 31-MAR-2020
 2 01-APR-2020 30-JUN-2020
 3 01-JUL-2020 30-SEP-2020
 4 01-OCT-2020 31-DEC-2020

```

DB2

Для создания четырех строк используем таблицу EMP и оконную функцию ROW_NUMBER OVER. Альтернативно, список строк можно создать с помощью оператора

WITH (как это и делается во многих решениях) или же выполнив запрос по любой таблице, содержащей как минимум четыре строки. Далее приводится решение на основе подхода с использованием функции ROW_NUMBER OVER:

```

1 select quarter(dy-1 day) QTR,
2     dy-3 month Q_start,
3     dy-1 day Q_end
4   from (
5 select (current_date -
6        (dayofyear(current_date)-1) day
7        + (rn*3) month) dy
8   from (
9 select row_number()over() rn
10  from emp
11  fetch first 4 rows only
12    ) x
13    ) y

```

Oracle

Для определения начальных и конечных дат каждого квартала используем функцию ADD_MONTHS, а для представления квартала, которому соответствуют определенные начальная и конечная дата, — функцию ROWNUM. Далее приводится решение с использованием таблицы EMP для создания четырех строк:

```

1 select rownum qtr,
2     add_months(trunc(sysdate, 'y'), (rownum-1)*3) q_start,
3     add_months(trunc(sysdate, 'y'), rownum*3)-1 q_end
4   from emp
5  where rownum <= 4

```

PostgreSQL

Сначала вычисляем первый день года на основании текущей даты, затем с помощью рекурсивного обобщенного табличного выражения определяем начальную дату для остальных трех кварталов, после чего вычисляем конечный день для каждого квартала:

```

with recursive x (dy, cnt)
  as (
select
    current_date - cast(extract(day from current_date) as integer) + 1 dy
    , id
  from t1
  union all
select cast(dy + interval '3 months' as date) , cnt+1
  from x
  where cnt+1 <= 4
)

```

```
select cast(dy - interval '3 months' as date) as Q_start
       , dy-1 as Q_end
       from x
```

MySQL

Определяем первый день года, а затем с помощью обобщенного табличного выражения создаем по одной строке для каждого квартала. Последний день каждого квартала вычисляем с помощью функции `ADDDATE` (три месяца после последнего дня предыдущего квартала или первый день следующего квартала минус 1):

```
1 with recursive x (dy,cnt)
2   as (
3 select
4     adddate(current_date,(-dayofyear(current_date))+1) dy
5     ,id
6   from t1
7  union all
8 select adddate(dy, interval 3 month ), cnt+1
9   from x
10  where cnt+1 <= 4
11 )
12
13 select quarter(adddate(dy,-1)) QTR
14           , date_add(dy, interval -3 month) Q_start
15           , adddate(dy,-1) Q_end
16   from x
17  order by 1;
```

SQL Server

С помощью рекурсивного оператора `WITH` создаем четыре строки, а затем с помощью функции `DATEADD` определяем начальные и конечные даты каждого квартала. Для представления квартала, которому соответствуют определенные начальная и конечная даты, используем функцию `DATEPART`:

```
1 with x (dy,cnt)
2   as (
3 select dateadd(d,-(datepart(dy,getdate())-1),getdate()),
4     1
5   from t1
6  union all
7 select dateadd(m,3,dy), cnt+1
8   from x
9  where cnt+1 <= 4
10 )
11 select datepart(q,dateadd(d,-1,dy)) QTR,
12        dateadd(m,-3,dy) Q_start,
13        dateadd(d,-1,dy) Q_end
```

```

14     from x
15     order by 1

```

Обсуждение

DB2

На первом шаге нужно создать для каждого квартала года по одной строке со значениями от 1 до 4. Эта задача выполняется с помощью вложенного запроса X, который, используя оконную функцию `ROW_NUMBER OVER` и оператор `FETCH FIRST`, возвращает четыре строки из таблицы EMP. Далее приводятся соответствующий запрос и его результаты:

```

select row_number() over () rn
       from emp
       fetch first 4 rows only

```

```

RN
--
1
2
3
4

```

На следующем шаге определяем первый день года и добавляем к нему *n* месяцев, где *n* — это трехкратное RN (к первому дню года добавляем 3, 6, 9 и 12 месяцев). Далее приводятся соответствующий запрос и его результаты:

```

select (current_date
       (dayofyear(current_date)-1) day
       + (rn*3) month) dy
       from (
select row_number() over () rn
       from emp
       fetch first 4 rows only
       ) x

```

```

DY
-----
01-APR-2005
01-JUL-2005
01-OCT-2005
01-JAN-2005

```

На этом этапе значения столбца `DY` на один день больше конечной даты каждого квартала. И поскольку нам нужно получить начальную и конечную даты для каждого квартала, из значений столбца `DY` вычитаем один день, получая конечные даты каждого квартала, и три месяца, получая начальные даты каждого квартала. Для представления квартала, которому соответствуют определенные начальная и ко-

нечная даты, используем функцию `QUARTER` со значением `DY-1` (конечная дата каждого квартала).

Oracle

Наличие встроенных функций `ROWNUM`, `TRUNC` и `ADD_MONTH` значительно облегчает решение этой задачи. Чтобы определить начальную дату каждого квартала, просто добавляем n месяцев к первому дню года, где n равно $(\text{ROWNUM}-1) \times 3$ (т. е. добавляем 0, 3, 6 и 9 месяцев). Чтобы определить конечную дату каждого квартала, добавляем n месяцев к первому дню года, где n равно $\text{ROWNUM} \times 3$, и вычитаем один день. Кстати, при работе с кварталами могут оказаться полезными функция `TO_CHAR` и/или функция `TRUNC` с опцией форматирования `Q`.

PostgreSQL, MySQL и SQL Server

Подобно некоторым предыдущим рецептам, решения для этих трех СУБД основаны на одном подходе, но используют разный синтаксис для операций с датами. На первом шаге определяем первый день года, после чего с помощью функции `DATEADD` или эквивалентной функции рекурсивно добавляем n месяцев, где n втрое больше текущей итерации. Таким образом, при общем количестве итераций, равном 4, добавляется 3×1 , 3×2 , 3×2 и 3×3 месяца. Далее приводятся соответствующий запрос и его результаты:

```
with x (dy,cnt)
    as (
select dateadd(d,-(datepart(dy,getdate())-1),getdate()),
       1
    from t1
    union all
select dateadd(m,3,dy), cnt+1
    from x
    where cnt+1 <= 4
)
select dy
    from x

DY
-----
01-APR-2020
01-JUL-2020
01-OCT-2020
01-JAN-2020
```

На этом этапе значения столбца `DY` на один день больше конечной даты каждого квартала. Чтобы получить конечную дату каждого квартала, просто вычитаем один день из значения `DY`, используя для этого функцию `DATEADD`. С помощью этой же функции определяем и начальную дату каждого квартала, вычитая три месяца из каждого значения `DY`. Для определения квартала, которому соответствуют опреде-

ленные начальная и конечная даты, используем функцию `DATEPART` (или ее эквивалент) с конечной датой каждого квартала. Пользователям PostgreSQL следует обратить внимание на необходимость приведения к типу (`CAST`) после добавления трех месяцев к начальной дате, чтобы обеспечить выравнивание типов данных. В противном случае типы данных будут разными, и оператор `UNION ALL` в рекурсивном обобщенном табличном выражении выдаст ошибку.

9.9. Определение начальной и конечной дат для заданного квартала

ЗАДАЧА

Для квартала года, заданного в формате `YYYYQ` (четыре цифры — год, одна цифра — квартал), требуется определить начальную и конечную даты.

РЕШЕНИЕ

Ключ к решению — найти квартал, используя функцию вычисления модуля (остатка) со значением `YYYYQ` в качестве параметра. (В качестве альтернативы модулю, поскольку формат года состоит из четырех цифр, вы можете просто извлечь последнюю цифру, чтобы получить квартал.) Полученное значение квартала просто умножаем на три, чтобы получить его последний месяц. В рассматриваемых решениях вложенный запрос `X` возвращает все четыре комбинации года и кварталов. Его результирующее множество выглядит таким образом:

```
select 20051 as yrq from t1 union all
select 20052 as yrq from t1 union all
select 20053 as yrq from t1 union all
select 20054 as yrq from t1
```

```
   YRQ
-----
20051
20052
20053
20054
```

DB2

Возвращаем год из вложенного запроса `X`, используя для этого функцию `SUBSTR`. Требуемый квартал вычисляем с помощью функции `MOD`:

```
1 select (q_end-2 month) q_start,
2        (q_end+1 month)-1 day q_end
3   from (
4 select date(substr(cast(yrq as char(4)),1,4) || '-'||
5        rtrim(cast(mod(yrq,10)*3 as char(2))) || '-1') q_end
```

```

6   from (
7   select 20051 yrq from t1 union all
8   select 20052 yrq from t1 union all
9   select 20053 yrq from t1 union all
10  select 20054 yrq from t1
11   ) x
12   ) y

```

Oracle

Возвращаем год из вложенного запроса X, используя для этого функцию SUBSTR. Требуемый квартал вычисляем с помощью функции MOD:

```

1   select add_months(q_end,-2) q_start,
2          last_day(q_end) q_end
3   from (
4   select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') q_end
5   from (
6   select 20051 yrq from dual union all
7   select 20052 yrq from dual union all
8   select 20053 yrq from dual union all
9   select 20054 yrq from dual
10   ) x
11   ) y

```

PostgreSQL

Возвращаем год из вложенного запроса X, используя для этого функцию SUBSTR. Требуемый квартал вычисляем с помощью функции MOD:

```

1   select date(q_end-(2*interval '1 month')) as q_start,
2          date(q_end+interval '1 month'-interval '1 day') as q_end
3   from (
4   select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') as q_end
5   from (
6   select 20051 as yrq from t1 union all
7   select 20052 as yrq from t1 union all
8   select 20053 as yrq from t1 union all
9   select 20054 as yrq from t1
10   ) x
11   ) y

```

MySQL

Возвращаем год из вложенного запроса X, используя для этого функцию SUBSTR. Требуемый квартал вычисляем с помощью функции MOD:

```

1   select date_add(
2          adddate(q_end,-day(q_end)+1),
3          interval -2 month) q_start,
4          q_end

```

```

5     from (
6 select last_day(
7     str_to_date(
8         concat(
9             substr(yrq,1,4),mod(yrq,10)*3),'%Y%m') q_end
10    from (
11 select 20051 as yrq from t1 union all
12 select 20052 as yrq from t1 union all
13 select 20053 as yrq from t1 union all
14 select 20054 as yrq from t1
15     ) x
16     ) y

```

SQL Server

Возвращаем год из вложенного запроса X, используя для этого функцию SUBSTRING. Требуемый квартал вычисляем с помощью оператора деления по модулю (%):

```

1 select dateadd(m,-2,q_end) q_start,
2     dateadd(d,-1,dateadd(m,1,q_end)) q_end
3     from (
4 select cast(substring(cast(yrq as varchar),1,4)+'-' +
5     cast(yrq%10*3 as varchar)+'-1' as datetime) q_end
6     from (
7 select 20051 as yrq from t1 union all
8 select 20052 as yrq from t1 union all
9 select 20052 as yrq from t1 union all
10 select 20054 as yrq from t1
11     ) x
12     ) y

```

Обсуждение

DB2

На первом шаге нужно найти год и квартал, с которыми будем работать. Год получаем, извлекая с помощью функции SUBSTR из вложенного запроса X (X.YRQ) подстроку года. Чтобы получить квартал, делим значение YRQ по модулю 10. Полученное значение квартала просто умножаем на три, чтобы получить его последний месяц. Далее приводятся соответствующий запрос и его результаты:

```

select substr(cast(yrq as char(4)),1,4) yr,
       mod(yrq,10)*3 mth
   from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
   ) x

```

YR	MTH
2005	3
2005	6
2005	9
2005	12

Используем полученные значения года и последнего месяца каждого квартала для формирования даты — в частности, первого дня последнего месяца каждого квартала. Для этого с помощью оператора конкатенации `||` объединяем год и месяц, а затем с помощью функции `DATE` преобразовываем полученную строку в дату:

```
select date(substr(cast(yrq as char(4)),1,4) || '-' ||
            rtrim(cast(mod(yrq,10)*3 as char(2))) || '-1') q_end
       from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
       ) x
```

```
Q_END
-----
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005
```

Значения `Q_END` представляют первый день последнего месяца каждого квартала. Чтобы получить последний день месяца, к `Q_END` добавляем один месяц, а затем вычитаем 1. А чтобы получить начальную дату каждого квартала, из `Q_END` вычитаем два месяца.

Oracle

На первом шаге нужно найти год и квартал, с которыми будем работать. Год получаем, извлекая с помощью функции `SUBSTR` из вложенного запроса `X (X.YRQ)` подстроку года. Чтобы получить квартал, делим значение `YRQ` по модулю 10. Полученное значение квартала просто умножаем на три, чтобы получить его последний месяц. Далее приводятся соответствующий запрос и его результаты:

```
select substr(yrq,1,4) yr, mod(yrq,10)*3 mth
       from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
       ) x
```


YR	MTH
2005	3
2005	6
2005	9
2005	12

Используем полученные значения года и последнего месяца каждого квартала для формирования даты — в частности, первого дня последнего месяца каждого квартала. Для этого с помощью оператора конкатенации `||` объединяем год и месяц, а затем с помощью функции `TO_DATE` преобразовываем полученную строку в дату:

```
select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') q_end
      from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
      ) x
```

```
Q_END
-----
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005
```

Значения `Q_END` представляют первый день последнего месяца каждого квартала. Применяя функцию `LAST_DAY` к этому значению, находим последний день месяца. А чтобы получить начальную дату каждого квартала, с помощью функции `ADD_MONTHS` вычитаем из `Q_END` два месяца.

PostgreSQL

На первом шаге нужно найти год и квартал, с которыми будем работать. Год получаем, извлекая с помощью функции `SUBSTR` из вложенного запроса `X (X.YRQ)` подстроку года. Чтобы получить квартал, делим значение `YRQ` по модулю 10. Полученное значение квартала просто умножаем на три, чтобы получить его последний месяц. Далее приводятся соответствующий запрос и его результаты:

```
select substr(yrq,1,4) yr, mod(yrq,10)*3 mth
      from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
      ) x
```

YR	MTH
2005	3
2005	6

```
2005      9
2005     12
```

Используем полученные значения года и последнего месяца каждого квартала для формирования даты — в частности, первого дня последнего месяца каждого квартала. Для этого с помощью оператора конкатенации `||` объединяем год и месяц, а затем с помощью функции `TO_DATE` преобразовываем полученную строку в дату:

```
select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') q_end
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
  ) x
```

```
Q_END
-----
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005
```

Значения `Q_END` представляют первый день последнего месяца каждого квартала. Чтобы получить последний день месяца, к `Q_END` добавляем один месяц, а затем вычитаем один день. А чтобы получить начальную дату каждого квартала, из `Q_END` вычитаем два месяца. Отображаем окончательные результаты как даты.

MySQL

На первом шаге нужно найти год и квартал, с которыми будем работать. Год получаем, извлекая с помощью функции `SUBSTR` из вложенного запроса `X (X.YRQ)` подстроку года. Чтобы получить квартал, делим значение `YRQ` по модулю 10. Полученное значение квартала просто умножаем на три, чтобы получить его последний месяц. Далее приводятся соответствующий запрос и его результаты:

```
select substr(yrq,1,4) yr, mod(yrq,10)*3 mth
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
  ) x
```

```
YR    MTH
----  ----
2005     3
2005     6
2005     9
2005    12
```

Используем полученные значения года и последнего месяца каждого квартала для формирования даты — в частности, последнего дня каждого квартала. Для этого с помощью функции `CONCAT` объединяем год и месяц, а затем с помощью функции `STR_TO_DATE` преобразовываем полученную строку в дату. Наконец, с помощью функции `LAST_DAY` определяем последний день каждого квартала:

```
select last_day(
    str_to_date(
        concat(
            substr(yrq,1,4),mod(yrq,10)*3), '%Y%m')
    ) q_end
from (
select 20051 as yrq from t1 union all
select 20052 as yrq from t1 union all
select 20053 as yrq from t1 union all
select 20054 as yrq from t1
) x
```

Q_END

```
-----
31-MAR-2005
30-JUN-2005
30-SEP-2005
31-DEC-2005
```

Поскольку у нас уже есть конечная дата каждого квартала, нам осталось вычислить только их начальные даты. С помощью функции `DAY` возвращаем день месяца, на который выпадает конец каждого квартала, и с помощью функции `ADDDATE` вычитаем полученное значение из значения `Q_END`, получая в результате последний день предыдущего месяца. Добавляя к полученному значению один день, получаем первый день последнего месяца каждого квартала. Наконец, посредством функции `DATE_ADD` отсчитываем два месяца от первого дня последнего месяца каждого квартала, получая их соответствующие начальные даты.

SQL Server

На первом шаге нужно найти год и квартал, с которыми будем работать. Год получаем, извлекая с помощью функции `SUBSTRING` из вложенного запроса `X (X.YRQ)` подстроку года. Чтобы получить квартал, делим значение `YRQ` по модулю 10. Полученное значение квартала просто умножаем на три, чтобы получить его последний месяц. Далее приводятся соответствующий запрос и его результаты:

```
select substring(yrq,1,4) yr, yrq%10*3 mth
from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
) x
```

YR	MTN
2005	3
2005	6
2005	9
2005	12

Используем полученные значения года и последнего месяца каждого квартала для формирования даты — в частности, первого дня последнего месяца каждого квартала. Для этого с помощью оператора конкатенации + объединяем год и месяц, а затем с помощью функции CAST преобразовываем полученную строку в дату:

```
select cast(substring(cast(yrq as varchar),1,4)+'-' +
cast(yrq%10*3 as varchar)+'-1' as datetime) q_end
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
  ) x
```

Q_END
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005

Значения Q_END представляют первый день последнего месяца каждого квартала. Чтобы получить первый день этого месяца, добавляем к Q_END один месяц и с помощью функции DATEADD вычитаем один день. Наконец, чтобы получить начальную дату каждого квартала, с помощью функции DATEADD вычитаем из Q_END два месяца.

9.10. Дополнение недостающих дат

ЗАДАЧА

Требуется создать строку для каждого дня (или недели, месяца, года) в пределах заданного временного интервала. Такие наборы строк часто используются при создании сводных отчетов. Например, нам нужно подсчитать количество принятых на работу служащих в каждом месяце каждого года, когда производился прием на работу. Выполнив соответствующую выборку по таблице EMP, можно видеть, что прием на работу производился с 2000 по 2003 г. включительно:

```
select distinct
  extract(year from hiredate) as year
  from emp
```

```

YEAR
-----
2000
2001
2002
2003

```

Мы хотим определить количество служащих, принятых на работу каждый месяц в этом интервале времени. Далее приводится часть требуемого результирующего множества:

MTH	NUM_HIRED
-----	-----
01-JAN-2001	0
01-FEB-2001	2
01-MAR-2001	0
01-APR-2001	1
01-MAY-2001	1
01-JUN-2001	1
01-JUL-2001	0
01-AUG-2001	0
01-SEP-2001	2
01-OCT-2001	0
01-NOV-2001	1
01-DEC-2001	2

РЕШЕНИЕ

В этом случае сложность состоит в том, чтобы возвращать строки даже для тех месяцев, в которых не было приемов на работу (т. е. со значением 0). Поскольку в период времени с 2000 года по 2003-й прием на работу осуществлялся не каждый месяц, нам надо самим сгенерировать эти пропущенные месяцы, а затем выполнить внешнее объединение полученных строк с таблицей EMP по столбцу HIREDATE (усекая фактические значения HIREDATE до месяца, чтобы их можно было сопоставлять со сгенерированными месяцами).

DB2

Генерируем все месяцы (точнее, первые дни каждого месяца с 1 января 2000 г. по 1 декабря 2003 г.) с помощью рекурсивного оператора WITH. Далее выполняем внешнее объединение полученных строк с таблицей EMP, а затем с помощью агрегатной функции COUNT подсчитываем количество приемов на работу для каждого месяца:

```

1 with x (start_date,end_date)
2   as (
3 select (min(hiredate)
4         dayofyear(min(hiredate)) day +1 day) start_date,
5        (max(hiredate)
6         dayofyear(max(hiredate)) day +1 day) +1 year end_date

```

```

7     from emp
8   union all
9   select start_date +1 month, end_date
10    from x
11   where (start_date +1 month) < end_date
12 )
13 select x.start_date mth, count(e.hiredate) num_hired
14    from x left join emp e
15         on (x.start_date = (e.hiredate-(day(hiredate)-1) day))
16   group by x.start_date
17  order by 1

```

Oracle

Генерируем все месяцы с 2000 г. по 2003 г. с помощью оператора `CONNECT`. Далее выполняем внешнее объединение полученных строк с таблицей `EMP`, а затем с помощью агрегатной функции `COUNT` подсчитываем количество приемов на работу для каждого месяца:

```

1   with x
2     as (
3   select add_months(start_date,level-1) start_date
4     from (
5   select min(trunc(hiredate,'y')) start_date,
6          add_months(max(trunc(hiredate,'y')),12) end_date
7     from emp
8          )
9   connect by level <= months_between(end_date,start_date)
10  )
11  select x.start_date MTH, count(e.hiredate) num_hired
12     from x left join emp e
13          on (x.start_date = trunc(e.hiredate,'mm'))
14   group by x.start_date
15  order by 1

```

PostgreSQL

Генерируем все месяцы, начиная с первого приема на работу, с помощью обобщенного табличного выражения. Затем выполняем внешнее левое объединение (`LEFT OUTER JOIN`) полученных строк с таблицей `EMP`, используя месяц и год каждой сгенерированной строки, чтобы включить подсчет (`COUNT`) количества приемов на работу за каждый месяц:

```

        with recursive x (start_date, end_date)
as
(
  select
    cast(min(hiredate) - (cast(extract(day from min(hiredate))

```

```

as integer) - 1) as date)
, max(hiredate)
from emp
union all
select cast(start_date + interval '1 month' as date)
, end_date
from x
where start_date < end_date
)

select x.start_date, count(hiredate)
from x left join emp
on (extract(month from start_date) =
    extract(month from emp.hiredate)
and extract(year from start_date)
= extract(year from emp.hiredate))
group by x.start_date
order by 1

```

MySQL

С помощью обобщенного табличного выражения генерируем все месяцы заданного периода, а затем выполняем внешнее объединение полученных строк с таблицей EMP и подсчитываем количество приемов на работу:

```

with recursive x (start_date, end_date)
as
(
select
    adddate(min(hiredate),
        -dayofyear(min(hiredate))+1) start_date
    , adddate(max(hiredate),
        -dayofyear(max(hiredate))+1) end_date
from emp
union all
select date_add(start_date, interval 1 month)
, end_date
from x
where date_add(start_date, interval 1 month) < end_date
)

select x.start_date mth, count(e.hiredate) num_hired
from x left join emp e
on (extract(year_month from start_date)
=
extract(year_month from e.hiredate))
group by x.start_date
order by 1;

```

SQL Server

Генерируем все месяцы (точнее, первые дни каждого месяца с 1 января 2000 г. по 1 декабря 2003 г.) с помощью рекурсивного оператора WITH. Далее выполняем внешнее объединение полученных строк с таблицей EMP, а затем с помощью агрегатной функции COUNT подсчитываем количество приемов на работу для каждого месяца:

```

1   with x (start_date,end_date)
2   as (
3   select (min(hiredate) -
4           datepart(dy,min(hiredate))+1) start_date,
5           dateadd(yy,1,
6                 (max(hiredate) -
7                 datepart(dy,max(hiredate))+1)) end_date
8   from emp
9   union all
10  select dateadd(mm,1,start_date), end_date
11  from x
12  where dateadd(mm,1,start_date) < end_date
13 )
14 select x.start_date mth, count(e.hiredate) num_hired
15  from x left join emp e
16  on (x.start_date =
17      dateadd(dd,-day(e.hiredate)+1,e.hiredate))
18  group by x.start_date
19  order by 1

```

Обсуждение

DB2

На первом шаге нужно сгенерировать все месяцы (вернее, первые дни каждого месяца) с 2000 г. по 2003 г. Начнем с нахождения всех граничных месяцев, используя для этого функцию DAYOFYEAR, передавая ей в качестве параметров результаты обработки значений HIREDATE функциями MIN и MAX:

```

select (min(hiredate)
        dayofyear(min(hiredate)) day +1 day) start_date,
       (max(hiredate)
        dayofyear(max(hiredate)) day +1 day) +1 year end_date
from emp

```

```

START_DATE  END_DATE
-----
01-JAN-2000 01-JAN-2004

```

Далее последовательно добавляем по одному месяцу к START_DATE, чтобы получить все месяцы, необходимые для конечного результирующего множества. Хотя значение END_DATE на один день больше, чем должно быть, это не является проблемой,

поскольку мы можем завершить процесс добавления месяцев к `START_DATE`, не включая `END_DATE` в результирующее множество. Соответствующий запрос и часть его результатов показаны далее:

```
with x (start_date,end_date)
  as (
select (min(hiredate)
        dayofyear(min(hiredate)) day +1 day) start_date,
        (max(hiredate)
        dayofyear(max(hiredate)) day +1 day) +1 year end_date
  from emp
 union all
select start_date +1 month, end_date
  from x
  where (start_date +1 month) < end_date
 )
select *
  from x
```

```
START_DATE END_DATE
-----
01-JAN-2000 01-JAN-2004
01-FEB-2000 01-JAN-2004
01-MAR-2000 01-JAN-2004
...
01-OCT-2003 01-JAN-2004
01-NOV-2003 01-JAN-2004
01-DEC-2003 01-JAN-2004
```

Получив таблицу со всеми требуемыми месяцами, можно просто выполнить ее внешнее объединение с таблицей `EMP` по столбцу `HIREDATE`. Поскольку для каждой даты `START_DATE` составляющая дня является первым числом месяца, значения столбца `EMP.HIREDATE` нужно усечь до первых дней соответствующих месяцев. Наконец, применяем к столбцу `EMP.HIREDATE` агрегатную функцию `COUNT`, чтобы подсчитать количество приемов на работу в каждом месяце.

Oracle

На первом шаге нужно сгенерировать все первые дни каждого месяца с 2000 г. по 2003 г. Начнем с нахождением всех граничных месяцев, применяя для этого функции `TRUNC` и `ADD_MONTHS` со значениями `MIN` и `MAX` к значениям `HIREDATE`:

```
select min(trunc(hiredate, 'y')) start_date,
        add_months(max(trunc(hiredate, 'y')),12) end_date
  from emp
```

```
START_DATE END_DATE
-----
01-JAN-2000 01-JAN-2004
```

Далее последовательно добавляем по одному месяцу к `START_DATE`, чтобы получить все месяцы, необходимые для конечного результирующего множества. Хотя значение `END_DATE` на один день больше действительного, это не проблема, поскольку мы можем завершить процесс добавления месяцев к `START_DATE`, не включая `END_DATE` в результирующее множество. Соответствующий запрос и часть его результатов показаны далее:

```
with x as (
select add_months(start_date,level-1) start_date
      from (
select min(trunc(hiredate,'y')) start_date,
       add_months(max(trunc(hiredate,'y')),12) end_date
      from emp
      )
connect by level <= months_between(end_date,start_date)
)
select *
      from x
```

```
START_DATE
-----
01-JAN-2000
01-FEB-2000
01-MAR-2000
...
01-OCT-2003
01-NOV-2003
01-DEC-2003
```

Получив таблицу со всеми требуемыми месяцами, можно просто выполнить ее внешнее объединение с таблицей `EMP` по столбцу `HIREDATE`. Поскольку для каждой даты `START_DATE` составляющая дня является первым числом месяца, значения столбца `EMP.HIREDATE` нужно усечь до первых дней соответствующих месяцев. Наконец, применяем к столбцу `EMP.HIREDATE` агрегатную функцию `COUNT`, чтобы подсчитать количество приемов на работу в каждом месяце.

PostgreSQL

Это решение для создания всех требуемых месяцев использует обобщенное табличное выражение и похоже на следующие далее решения для `MySQL` и `SQL Server`. На первом шаге, как обычно, генерируются граничные даты с помощью агрегатных функций. Для вычисления первой и последней дат приема на работу можно было бы просто применить функции `MIN` и `MAX`, но результаты будут более понятны, если определить первое число месяца, в котором произошел первый прием на работу.

MySQL

На первом шаге находим граничные даты с помощью агрегатных функций `MIN` и `MAX` в комбинации с функциями `DAYOFYEAR` и `ADDDATE`. Далее приводится результирующее множество, возвращаемое вложенным запросом `X`:

```
with recursive x (start_date,end_date)
    as (
    select
        adddate(min(hiredate) ,
        -dayofyear(min(hiredate))+1) start_date
        ,adddate(max(hiredate) ,
        -dayofyear(max(hiredate))+1) end_date
    from emp
    union all
    select date_add(start_date,interval 1 month)
        , end_date
    from x
    where date_add(start_date, interval 1 month) < end_date
    )
select * from x

select adddate(min(hiredate) ,-dayofyear(min(hiredate))+1) min_hd,
       adddate(max(hiredate) ,-dayofyear(max(hiredate))+1) max_hd
from emp
```

```
MIN_HD      MAX_HD
-----
01-JAN-2000 01-JAN-2003
```

Получив таблицу со всеми требуемыми месяцами, можно просто выполнить ее внешнее объединение с таблицей `EMP` по столбцу `HIREDATE`. Поскольку для каждой даты `START_DATE` составляющая дня является первым числом месяца, значения столбца `EMP.HIREDATE` нужно усечь до первых дней соответствующих месяцев. Наконец, применяем к столбцу `EMP.HIREDATE` агрегатную функцию `COUNT`, чтобы подсчитать количество приемов на работу в каждом месяце.

SQL Server

На первом шаге генерируем все месяцы (вернее, первые дни каждого месяца) с 2000 г. по 2003 г. Затем определяем граничные месяцы, применяя функцию `DAYOFYEAR` к значениям `HIREDATE`, возвращаемым функциями `MIN` и `MAX`:

```
select (min(hiredate) -
       datepart(dy,min(hiredate))+1) start_date,
       dateadd(yy,1,
       (max(hiredate) -
       datepart(dy,max(hiredate))+1)) end_date
from emp
```

```
START_DATE END_DATE
-----
01-JAN-2000 01-JAN-2004
```

Далее последовательно добавляем по одному месяцу к `START_DATE`, чтобы получить все месяцы, необходимые для конечного результирующего множества. Хотя значение `END_DATE` на один день больше действительного, это не проблема, поскольку мы можем завершить процесс добавления месяцев к `START_DATE`, не включая `END_DATE` в результирующее множество. Далее приводится соответствующий запрос и часть возвращаемых им данных:

```
with x (start_date,end_date)
  as (
select (min(hiredate) -
       datepart(dy,min(hiredate))+1) start_date,
       dateadd(yy,1,
              (max(hiredate) -
               datepart(dy,max(hiredate))+1)) end_date
  from emp
 union all
select dateadd(mm,1,start_date), end_date
  from x
 where dateadd(mm,1,start_date) < end_date
 )
select *
  from x
```

```
START_DATE END_DATE
-----
01-JAN-2000 01-JAN-2004
01-FEB-2000 01-JAN-2004
01-MAR-2000 01-JAN-2004
...
01-OCT-2003 01-JAN-2004
01-NOV-2003 01-JAN-2004
01-DEC-2003 01-JAN-2004
```

Получив таблицу со всеми требуемыми месяцами, можно просто выполнить ее внешнее объединение с таблицей `EMP` по столбцу `HIREDATE`. Поскольку для каждой даты `START_DATE` составляющая дня является первым числом месяца, значения столбца `EMP.HIREDATE` нужно усечь до первых дней соответствующих месяцев. Наконец, применяем к столбцу `EMP.HIREDATE` агрегатную функцию `COUNT`, чтобы подсчитать количество приемов на работу в каждом месяце.

9.11. Поиск по заданным единицам времени

ЗАДАЧА

Требуется найти даты, содержащие заданный месяц, день недели или какую-либо другую единицу времени. Например, надо найти всех служащих, принятых на работу в феврале или декабре, а также служащих, принятых на работу во вторник.

РЕШЕНИЕ

Названия месяцев и дней недели дат определяем с помощью встроенных функций используемой СУБД. Этот рецепт может пригодиться во многих случаях. Например, для поиска по части значения `hiredate` (месяцу или какой-либо другой составляющей), игнорируя при этом все другие составляющие даты. Примеры решений этой задачи выполняют поиск по месяцу и дню недели. Разобравшись с работой функций форматирования дат своей СУБД, вы сможете с легкостью подкорректировать эти решения для поиска дат по году, кварталу, комбинации года и квартала или месяца и года и т. д.

DB2 и MySQL

Чтобы найти месяц и день недели приема служащего на работу, используем функции `MONTHNAME` и `DAYNAME`, соответственно:

```
1 select ename
2   from emp
3 where monthname(hiredate) in ('February','December')
4    or dayname(hiredate) = 'Tuesday'
```

Oracle и PostgreSQL

Месяц и день недели приема служащего на работу определяем с помощью функции `TO_CHAR`, а затем с помощью функции `RTRIM` удаляем замыкающие пробелы.

```
1 select ename
2   from emp
3 where rtrim(to_char(hiredate,'month')) in ('february','december')
4    or rtrim(to_char(hiredate,'day')) = 'tuesday'
```

SQL Server

Месяц и день недели приема служащего на работу определяем с помощью функции `DATENAME`:

```
1 select ename
2   from emp
3 where datename(m,hiredate) in ('February','December')
4    or datename(dw,hiredate) = 'Tuesday'
```

Обсуждение

Ключевой момент каждого решения — просто знать, какие функции использовать и как с ними работать. Чтобы проверить возвращаемые этими функциями значения, помещаем их в выражение `SELECT` и смотрим, что получаем на выходе. Далее приводится пример такого запроса (составленного на основе синтаксиса SQL Server) для служащих отдела 10 и его результирующее множество:

```
select ename,datename(m,hiredate) mth,datename(dw,hiredate) dw
       from emp
       where deptno = 10
```

ENAME	MTH	DW
CLARK	June	Tuesday
KING	November	Tuesday
MILLER	January	Saturday

Выяснив, какие данные возвращаются функцией или функциями в каждом решении, использовать их для нахождения строк не представляет большого труда.

9.12. Сравнение записей по определенным частям даты

ЗАДАЧА

Требуется определить служащих, которые были приняты на работу в один и тот же месяц и день недели. Например, служащие, принятые на работу, скажем, в понедельник 10 марта 2008 г. и в понедельник 2 марта 2001 г., должны быть включены в результаты поиска из-за совпадения месяца и дня недели даты их приема на работу. В таблице `EMP` только трое служащих соответствуют этому требованию. То есть результирующее множество должно иметь следующий вид:

```
MSG
-----
JAMES was hired on the same month and weekday as FORD
SCOTT was hired on the same month and weekday as JAMES
SCOTT was hired on the same month and weekday as FORD
```

РЕШЕНИЕ

Так как мы хотим сравнивать значение `HIREDATE` одного служащего со значениями `HIREDATE` других служащих, нам нужно выполнить самообъединение таблицы `EMP`. Так мы получим для сравнения все возможные комбинации значений `HIREDATE`. Затем надо будет просто извлечь день недели и месяц из каждого значения `HIREDATE` и сравнить их.

DB2

Выполнив самообъединение таблицы EMP, используем функцию DAYOFWEEK для возвращения числового значения дня недели. Для возвращения названия месяца применим функцию MONTHNAME:

```
1 select a.ename ||
2     ' was hired on the same month and weekday as '||
3     b.ename msg
4   from emp a, emp b
5  where (dayofweek(a.hiredate),monthname(a.hiredate)) =
6        (dayofweek(b.hiredate),monthname(b.hiredate))
7     and a.empno < b.empno
8   order by a.ename
```

Oracle и PostgreSQL

Выполнив самообъединение таблицы EMP, используем функцию TO_CHAR для извлечения из значений HIREDATE дня недели и названия месяца для последующего сравнения:

```
1 select a.ename ||
2     ' was hired on the same month and weekday as '||
3     b.ename as msg
4   from emp a, emp b
5  where to_char(a.hiredate,'DMON') =
6        to_char(b.hiredate,'DMON')
7     and a.empno < b.empno
8   order by a.ename
```

MySQL

Выполнив самообъединение таблицы EMP, используем функцию DATE_FORMAT для извлечения из значений HIREDATE дня недели и названия месяца для последующего сравнения:

```
1 select concat(a.ename,
2     ' was hired on the same month and weekday as ',
3     b.ename) msg
4   from emp a, emp b
5  where date_format(a.hiredate,'%w%M') =
6        date_format(b.hiredate,'%w%M')
7     and a.empno < b.empno
8   order by a.ename
```

SQL Server

Выполнив самообъединение таблицы EMP, используем функцию DATENAME для извлечения из значений HIREDATE дня недели и названия месяца для последующего сравнения:

```

1 select a.ename +
2     ' was hired on the same month and weekday as '+
3     b.ename msg
4     from emp a, emp b
5     where datename(dw,a.hiredate) = datename(dw,b.hiredate)
6         and datename(m,a.hiredate) = datename(m,b.hiredate)
7         and a.empno < b.empno
8     order by a.ename

```

Обсуждение

Единственная разница между этими решениями — разные функции, используемые для форматирования даты значения `HIREDATE`. В этом обсуждении мы будем ссылаться на решение для Oracle и PostgreSQL (т. к. оно самое короткое), но приводимые объяснения также действительны и для других решений.

На первом шаге выполняем самообъединение таблицы `EMP`, чтобы можно было сравнить значение `HIREDATE` каждого служащего со значениями `HIREDATE` всех других служащих. Рассмотрим приведенные далее результаты запроса (отфильтрованные по служащему `SCOTT`):

```

select a.ename as scott, a.hiredate as scott_hd,
       b.ename as other_emps, b.hiredate as other_hds
   from emp a, emp b
  where a.ename = 'SCOTT'
     and a.empno != b.empno

```

SCOTT	SCOTT_HD	OTHER_EMPS	OTHER_HDS
SCOTT	09-DEC-2002	SMITH	17-DEC-2000
SCOTT	09-DEC-2002	ALLEN	20-FEB-2001
SCOTT	09-DEC-2002	WARD	22-FEB-2001
SCOTT	09-DEC-2002	JONES	02-APR-2001
SCOTT	09-DEC-2002	MARTIN	28-SEP-2001
SCOTT	09-DEC-2002	BLAKE	01-MAY-2001
SCOTT	09-DEC-2002	CLARK	09-JUN-2001
SCOTT	09-DEC-2002	KING	17-NOV-2001
SCOTT	09-DEC-2002	TURNER	08-SEP-2001
SCOTT	09-DEC-2002	ADAMS	12-JAN-2003
SCOTT	09-DEC-2002	JAMES	03-DEC-2001
SCOTT	09-DEC-2002	FORD	03-DEC-2001
SCOTT	09-DEC-2002	MILLER	23-JAN-2002

Самообъединение таблицы `EMP` дает нам возможность сравнивать значение `HIREDATE` служащего `SCOTT` со значениями `HIREDATE` всех других служащих. Фильтрация выполняется по столбцу `EMPNO`, чтобы значение `HIREDATE` для служащего `SCOTT` не возвращалось в столбце `OTHER_HDS` дат приема на работу других служащих. На следующем шаге применяем встроенную функцию или функции используемой СУБД

для сравнения дня недели и названия месяца значений `HIREDATE`, выбирая только те значения, которые соответствуют заданному:

```
select a.ename as emp1, a.hiredate as emp1_hd,
       b.ename as emp2, b.hiredate as emp2_hd
from emp a, emp b
where to_char(a.hiredate, 'DMON') =
      to_char(b.hiredate, 'DMON')
and a.empno != b.empno
order by 1
```

EMP1	EMP1_HD	EMP2	EMP2_HD
FORD	03-DEC-2001	SCOTT	09-DEC-2002
FORD	03-DEC-2001	JAMES	03-DEC-2001
JAMES	03-DEC-2001	SCOTT	09-DEC-2002
JAMES	03-DEC-2001	FORD	03-DEC-2001
SCOTT	09-DEC-2002	JAMES	03-DEC-2001
SCOTT	09-DEC-2002	FORD	03-DEC-2001

На этом этапе сопоставлены все правильные значения `HIREDATE`, но результирующее множество содержит шесть строк, а не три, как показано в *разд. «Задача»* этого рецепта. Лишние строки объясняются фильтрацией по столбцу `EMPNO`. Оператор `!=` («не равно») не обеспечивает удаления обратных равенств. Например, служащие `FORD` и `SCOTT` сопоставлены как в первой строке, так и в последней, только в обратном порядке: `SCOTT` и `FORD`. Шесть строк результирующего множества технически правильны, но избыточны. Чтобы избавиться от лишних строк, нужно использовать оператор `<` («меньше, чем») — значения `HIREDATE` удаляются, чтобы приблизить результаты промежуточных запросов к результирующему множеству:

```
select a.ename as emp1, b.ename as emp2
from emp a, emp b
where to_char(a.hiredate, 'DMON') =
      to_char(b.hiredate, 'DMON')
and a.empno < b.empno
order by 1
```

EMP1	EMP2
JAMES	FORD
SCOTT	JAMES
SCOTT	ORD

На последнем шаге выполняется простая конкатенация составных частей результирующего множества, чтобы сформировать сообщение.

9.13. Выявление наложений диапазонов дат

ЗАДАЧА

Требуется выявить всех служащих, начинающих новый проект до завершения текущего. Рассмотрим, например, таблицу EMP_PROJECT:

```
select *
  from emp_project
```

EMPNO	ENAME	PROJ_ID	PROJ_START	PROJ_END
7782	CLARK	1	16-JUN-2005	18-JUN-2005
7782	CLARK	4	19-JUN-2005	24-JUN-2005
7782	CLARK	7	22-JUN-2005	25-JUN-2005
7782	CLARK	10	25-JUN-2005	28-JUN-2005
7782	CLARK	13	28-JUN-2005	02-JUL-2005
7839	KING	2	17-JUN-2005	21-JUN-2005
7839	KING	8	23-JUN-2005	25-JUN-2005
7839	KING	14	29-JUN-2005	30-JUN-2005
7839	KING	11	26-JUN-2005	27-JUN-2005
7839	KING	5	20-JUN-2005	24-JUN-2005
7934	MILLER	3	18-JUN-2005	22-JUN-2005
7934	MILLER	12	27-JUN-2005	28-JUN-2005
7934	MILLER	15	30-JUN-2005	03-JUL-2005
7934	MILLER	9	24-JUN-2005	27-JUN-2005
7934	MILLER	6	21-JUN-2005	23-JUN-2005

Здесь можно видеть, что служащий KING начал работать над проектом PROJ_ID 8 до завершения проекта PROJ_ID 5, а также начал работать над проектом PROJ_ID 5 до завершения проекта PROJ_ID 2. Таким образом, результирующее множество должно иметь следующий вид:

EMPNO	ENAME	MSG
7782	CLARK	project 7 overlaps project 4
7782	CLARK	project 10 overlaps project 7
7782	CLARK	project 13 overlaps project 10
7839	KING	project 8 overlaps project 5
7839	KING	project 5 overlaps project 2
7934	MILLER	project 12 overlaps project 9
7934	MILLER	project 6 overlaps project 3

РЕШЕНИЕ

Ключ к решению — найти те строки, в которых дата начала PROJ_START одного проекта находится между датами начала (PROJ_START) и окончания (PROJ_END) другого проекта включительно. На первом шаге нужно получить возможность сравнивать дату каждого проекта служащего с датами всех его других проектов. Эту задачу

решим, выполнив самообъединение таблицы EMP_PROJECT по столбцу ENAME, создав, таким образом, все возможные комбинации двух проектов для каждого служащего. Чтобы выявить наложения, просто находим строки, в которых значение PROJ_START для любого значения PROJ_ID попадает между значениями PROJ_START и PROJ_END для другого PROJ_ID того же самого служащего.

DB2, PostgreSQL и Oracle

Сначала выполняем самообъединение таблицы EMP_PROJECT. Затем с помощью оператора конкатенации || формируем сообщение, описывающее перекрывающиеся проекты:

```

1 select a.empno,a.ename,
2         'project '||b.proj_id||
3         ' overlaps project '||a.proj_id as msg
4   from emp_project a,
5        emp_project b
6  where a.empno = b.empno
7        and b.proj_start >= a.proj_start
8        and b.proj_start <= a.proj_end
9        and a.proj_id != b.proj_id

```

MySQL

Сначала выполняем самообъединение таблицы EMP_PROJECT. Затем с помощью функции конкатенации CONCAT формируем сообщение, описывающее перекрывающиеся проекты:

```

1 select a.empno,a.ename,
2        concat('project ',b.proj_id,
3              ' overlaps project ',a.proj_id) as msg
4   from emp_project a,
5        emp_project b
6  where a.empno = b.empno
7        and b.proj_start >= a.proj_start
8        and b.proj_start <= a.proj_end
9        and a.proj_id != b.proj_id

```

SQL Server

Сначала выполняем самообъединение таблицы EMP_PROJECT. Затем с помощью оператора конкатенации + формируем сообщение, описывающее перекрывающиеся проекты:

```

1 select a.empno,a.ename,
2        'project '+b.proj_id+
3        ' overlaps project '+a.proj_id as msg
4   from emp_project a,
5        emp_project b

```

```

6  where a.empno = b.empno
7     and b.proj_start >= a.proj_start
8     and b.proj_start <= a.proj_end
9     nd a.proj_id != b.proj_id

```

Обсуждение

Решения различаются между собой только способами конкатенации строк, поэтому рассмотрим их все на основе синтаксиса решения для DB2. На первом шаге выполняем самообъединение таблицы EMP_PROJECT, чтобы получить возможность сравнения дат PROJ_START для разных проектов. Далее приводится результат самообъединения для служащего KING. Он показывает, как каждый проект «видит» другие проекты:

```

select a.ename,
       a.proj_id as a_id,
       a.proj_start as a_start,
       a.proj_end as a_end,
       b.proj_id as b_id,
       b.proj_start as b_start
from emp_project a,
     emp_project b
where a.ename = 'KING'
     and a.empno = b.empno
     and a.proj_id != b.proj_id
order by 2

```

ENAME	A_ID	A_START	A_END	B_ID	B_START
KING	2	17-JUN-2005	21-JUN-2005	8	23-JUN-2005
KING	2	17-JUN-2005	21-JUN-2005	14	29-JUN-2005
KING	2	17-JUN-2005	21-JUN-2005	11	26-JUN-2005
KING	2	17-JUN-2005	21-JUN-2005	5	20-JUN-2005
KING	5	20-JUN-2005	24-JUN-2005	2	17-JUN-2005
KING	5	20-JUN-2005	24-JUN-2005	8	23-JUN-2005
KING	5	20-JUN-2005	24-JUN-2005	11	26-JUN-2005
KING	5	20-JUN-2005	24-JUN-2005	14	29-JUN-2005
KING	8	23-JUN-2005	25-JUN-2005	2	17-JUN-2005
KING	8	23-JUN-2005	25-JUN-2005	14	29-JUN-2005
KING	8	23-JUN-2005	25-JUN-2005	5	20-JUN-2005
KING	8	23-JUN-2005	25-JUN-2005	11	26-JUN-2005
KING	11	26-JUN-2005	27-JUN-2005	2	17-JUN-2005
KING	11	26-JUN-2005	27-JUN-2005	8	23-JUN-2005
KING	11	26-JUN-2005	27-JUN-2005	14	29-JUN-2005
KING	11	26-JUN-2005	27-JUN-2005	5	20-JUN-2005
KING	14	29-JUN-2005	30-JUN-2005	2	17-JUN-2005
KING	14	29-JUN-2005	30-JUN-2005	8	23-JUN-2005
KING	14	29-JUN-2005	30-JUN-2005	5	20-JUN-2005
KING	14	29-JUN-2005	30-JUN-2005	11	26-JUN-2005

Как можно видеть по содержимому результирующего множества, самообъединение значительно облегчает задачу выявления накладывающихся дат: просто возвращаем каждую строку, в которой дата `B_START` находится между датами `A_START` и `A_END`. Предикат `WHERE` в строках 7 и 8 решения делает именно это:

```
and b.proj_start >= a.proj_start
and b.proj_start <= a.proj_end
```

Получив требуемые строки, формируем сообщения, просто конкатенируя возвращенные значения.

Если каждый служащий занимается фиксированным количеством проектов, пользователи Oracle могут воспользоваться оконной функцией `LEAD OVER`, чтобы избежать необходимости выполнять самообъединение. Это может быть удобно в случае, если выполнять самообъединение слишком накладно для конкретного рассматриваемого случая. Например, если оно требует больше ресурсов, чем сортировки, требуемые для функции `LEAD OVER`. Рассмотрим, например, альтернативное решение с использованием функции `LEAD OVER` для служащего `KING`:

```
select empno,
       ename,
       proj_id,
       proj_start,
       proj_end,
       case
when lead(proj_start,1)over(order by proj_start)
   between proj_start and proj_end
then lead(proj_id)over(order by proj_start)
when lead(proj_start,2)over(order by proj_start)
   between proj_start and proj_end
then lead(proj_id)over(order by proj_start)
when lead(proj_start,3)over(order by proj_start)
   between proj_start and proj_end
then lead(proj_id)over(order by proj_start)
when lead(proj_start,4)over(order by proj_start)
   between proj_start and proj_end
then lead(proj_id)over(order by proj_start)
end is_overlap
from emp_project
where ename = 'KING'
```

EMPNO	ENAME	PROJ_ID	PROJ_START	PROJ_END	IS_OVERLAP
7839	KING	2	17-JUN-2005	21-JUN-2005	5
7839	KING	5	20-JUN-2005	24-JUN-2005	8
7839	KING	8	23-JUN-2005	25-JUN-2005	
7839	KING	11	26-JUN-2005	27-JUN-2005	
7839	KING	14	29-JUN-2005	30-JUN-2005	

Поскольку для служащего KING количество проектов фиксировано и равно пяти, даты всех проектов можно сравнить, не прибегая к самообъединению, а с помощью функции LEAD OVER. После этого формирование конечного результирующего множества не представляет никаких проблем. Просто выбираем те строки, в которых значение IS_OVERLAP не равно NULL:

```
select empno,ename,
       'project '||is_overlap||
       ' overlaps project '||proj_id msg
from (
select empno,
       ename,
       proj_id,
       proj_start,
       proj_end,
       case
when lead(proj_start,1)over(order by proj_start)
  between proj_start and proj_end
then lead(proj_id)over(order by proj_start)
when lead(proj_start,2)over(order by proj_start)
  between proj_start and proj_end
then lead(proj_id)over(order by proj_start)
when lead(proj_start,3)over(order by proj_start)
  between proj_start and proj_end
then lead(proj_id)over(order by proj_start)
when lead(proj_start,4)over(order by proj_start)
  between proj_start and proj_end
then lead(proj_id)over(order by proj_start)
end is_overlap
from emp_project
where ename = 'KING'
)
where is_overlap is not null
```

```
EMPNO ENAME MSG
-----
7839 KING project 5 overlaps project 2
7839 KING project 8 overlaps project 5
```

Чтобы решение работало со всеми служащими, а не только со служащим KING, в функции LEAD OVER выполняем сегментацию по ENAME:

```
select empno,ename,
       'project '||is_overlap||
       ' overlaps project '||proj_id msg
from (
select empno,
       ename,
       proj_id,
```

```

proj_start,
proj_end,
case
when lead(proj_start,1)over(partition by ename
                           order by proj_start)
   between proj_start and proj_end
then lead(proj_id)over(partition by ename
                       order by proj_start)
when lead(proj_start,2)over(partition by ename
                             order by proj_start)
   between proj_start and proj_end
then lead(proj_id)over(partition by ename
                       order by proj_start)
when lead(proj_start,3)over(partition by ename
                             order by proj_start)
   between proj_start and proj_end
then lead(proj_id)over(partition by ename
                       order by proj_start)
when lead(proj_start,4)over(partition by ename
                             order by proj_start)
   between proj_start and proj_end
then lead(proj_id)over(partition by ename
                       order by proj_start)

end is_overlap
from emp_project
)
where is_overlap is not null

```

```

EMPNO ENAME MSG
-----
7782 CLARK project 7 overlaps project 4
7782 CLARK project 10 overlaps project 7
7782 CLARK project 13 overlaps project 10
7839 KING project 5 overlaps project 2
7839 KING project 8 overlaps project 5
7934 MILLER project 6 overlaps project 3
7934 MILLER project 12 overlaps project 9

```

9.14. Подведем итоги

Любому пользователю баз данных часто приходится сталкиваться с задачей работы датами — последовательность событий, сохраненная вместе со своими датами, побуждает бизнес-пользователей задавать самые разные вопросы. В то же время сфера работы с датами менее всего стандартизована в СУБД разных разработчиков. Мы надеемся, что в этой главе мы смогли убедительно продемонстрировать, что даже при использовании в запросах для работы с датами разных синтаксисов все они могут быть основаны на общей логике.

Работа с диапазонами значений

В этой главе рассматриваются «рутинные» запросы по диапазонам данных. Диапазоны встречаются в повседневной жизни на каждом шагу. Например, работа над проектом занимает диапазон последовательных интервалов времени. В SQL часто приходится выполнять поиск или генерировать диапазоны значений, или иным образом манипулировать диапазонами данных. Уровень сложности рассматриваемых в этой главе запросов несколько выше по сравнению с запросами, приведенными в предыдущих главах, но они так же широко употребляются и помогут вам понять, каким мощным средством может быть SQL для изучивших его в достаточном объеме.

10.1. Поиск диапазона последовательных значений

ЗАДАЧА

Требуется определить диапазон строк, представляющих ряд последовательных проектов. Например, рассмотрим следующее представление V, содержащее данные о ряде проектов и их начальные и конечные даты:

```
select *  
  from V
```

```
PROJ_ID PROJ_START  PROJ_END  
-----  
  1 01-JAN-2020 02-JAN-2020  
  2 02-JAN-2020 03-JAN-2020  
  3 03-JAN-2020 04-JAN-2020  
  4 04-JAN-2020 05-JAN-2020  
  5 06-JAN-2020 07-JAN-2020  
  6 16-JAN-2020 17-JAN-2020  
  7 17-JAN-2020 18-JAN-2020  
  8 18-JAN-2020 19-JAN-2020  
  9 19-JAN-2020 20-JAN-2020  
 10 21-JAN-2020 22-JAN-2020  
 11 26-JAN-2020 27-JAN-2020  
 12 27-JAN-2020 28-JAN-2020  
 13 28-JAN-2020 29-JAN-2020  
 14 29-JAN-2020 30-JAN-2020
```


За исключением первой строки, значение `PROJ_START` каждой строки должно равняться значению `PROJ_END` предшествующей ей строки (которая для текущей строки определяется, как строка со значением `PROJ_ID - 1`). Рассмотрев первые пять строк представления `V`, можно заметить, что проекты со значениями `PROJ_ID` с 1 по 3 являются членами одной «группы», поскольку значение `PROJ_END` каждого из них равно значению `PROJ_START` следующего проекта. Так как мы хотим найти интервал дат для последовательных проектов, нам нужно вернуть все строки, в которых значение `PROJ_END` текущей строки равно значению `PROJ_START` следующей за ней строки. То есть, если бы исходное результирующее множество состояло только из первых пяти строк, нам нужно было бы вернуть только первые три строки. А для всех 14 строк представления `V` конечное результирующее множество должно быть следующим:

PROJ_ID	PROJ_START	PROJ_END
1	01-JAN-2020	02-JAN-2020
2	02-JAN-2020	03-JAN-2020
3	03-JAN-2020	04-JAN-2020
6	16-JAN-2020	17-JAN-2020
7	17-JAN-2020	18-JAN-2020
8	18-JAN-2020	19-JAN-2020
11	26-JAN-2020	27-JAN-2020
12	27-JAN-2020	28-JAN-2020
13	28-JAN-2020	29-JAN-2020

Строки со значениями `PROJ_ID` 4, 5, 9, 10 и 14 исключены из этого результирующего множества, поскольку значение `PROJ_END` каждой из этих строк не совпадает со значением `PROJ_START` следующей за ней строки.

РЕШЕНИЕ

Решение в полном объеме использует оконную функцию `LEAD OVER` для проверки значения `PROJ_START` строки и тем самым избегает выполнения самообъединения, требуемого до введения оконных функций:

```

1 select proj_id, proj_start, proj_end
2     from (
3 select proj_id, proj_start, proj_end,
4         lead(proj_start)over(order by proj_id) next_proj_start
5     from V
6         ) alias
7  where next_proj_start = proj_end

```

Обсуждение

Предлагаемое решение пригодно для всех видов СУБД, работа с которыми рассматривается в этой книге: DB2, MySQL, PostgreSQL, SQL Server и Oracle. Хотя можно разработать решение на основе самообъединения, для такого типа задач

идеально подходит функция `LEAD OVER`, использование которой к тому же и более интуитивно. Функция `LEAD OVER` позволяет проверять другие строки без необходимости выполнять самообъединение (хотя для этого функция должна упорядочить исходное результирующее множество). Рассмотрим результаты выборки вложенным запросом (строки 3–5) строк со значениями `ID` 1 и 4:

```
select *
  from (
select proj_id, proj_start, proj_end,
       lead(proj_start)over(order by proj_id) next_proj_start
  from v
  )
 where proj_id in ( 1, 4 )
```

PROJ_ID	PROJ_START	PROJ_END	NEXT_PROJ_START
1	01-JAN-2020	02-JAN-2020	02-JAN-2020
4	04-JAN-2020	05-JAN-2020	06-JAN-2020

Изучив этот фрагмент кода и его результирующее множество, можно очень легко понять, почему строка со значением `PROJ_ID` 4 не попала в конечное результирующее множество полного решения. А не попала она по той причине, что ее дата `05-JAN-2020` столбца `PROJ_END` не совпадает с датой `06-JAN-2020` столбца `PROJ_START` следующего проекта.

Функция `LEAD OVER` чрезвычайно полезна в случае задач, подобных этой, особенно при исследовании частичных результатов. При работе с оконными функциями следует помнить, что они выполняются после операторов `FROM` и `WHERE`, поэтому в приведенном запросе функцию `LEAD OVER` необходимо вставить во вложенный запрос. В противном случае она применяется к результирующему множеству после того, как оператор `WHERE` отфильтрует все строки за исключением строк со значениями `PROJ_ID`, равными 1 и 4.

Теперь, в зависимости от того, как рассматриваются данные, строку с `PROJ_ID` 4 может потребоваться включить в конечное результирующее множество. Рассмотрим первые пять строк представления `V`:

```
select *
  from v
 where proj_id <= 5
```

PROJ_ID	PROJ_START	PROJ_END
1	01-JAN-2020	02-JAN-2020
2	02-JAN-2020	03-JAN-2020
3	03-JAN-2020	04-JAN-2020
4	04-JAN-2020	05-JAN-2020
5	06-JAN-2020	07-JAN-2020

Если согласно заданным требованиям проект `PROJ_ID` 4 считается смежным (поскольку его дата `PROJ_START` совпадает с датой `PROJ_END` проекта `PROJ_ID` 3) и отсеи-

ваться должен только проект PROJ_ID 5, то предлагаемое решение является неправильным (!) или, по крайней мере, неполным:

```
select proj_id, proj_start, proj_end
       from (
select proj_id, proj_start, proj_end,
       lead(proj_start)over(order by proj_id) next_start
       from V
       where proj_id <= 5
       )
       where proj_end = next_start
```

```
PROJ_ID PROJ_START PROJ_END
-----
1 01-JAN-2020 02-JAN-2020
2 02-JAN-2020 03-JAN-2020
3 03-JAN-2020 04-JAN-2020
```

Если проект PROJ_ID 4 должен быть включен, нужно просто добавить в запрос функцию LAG OVER и применить дополнительный фильтр в операторе WHERE:

```
select proj_id, proj_start, proj_end
       from (
select proj_id, proj_start, proj_end,
       lead(proj_start)over(order by proj_id) next_start,
       lag(proj_end)over(order by proj_id) last_end
       from V
       where proj_id <= 5
       )
       where proj_end = next_start
            or proj_start = last_end
```

```
PROJ_ID PROJ_START PROJ_END
-----
1 01-JAN-2020 02-JAN-2020
2 02-JAN-2020 03-JAN-2020
3 03-JAN-2020 04-JAN-2020
4 04-JAN-2020 05-JAN-2020
```

Теперь проект PROJ_ID 4 входит в конечное результирующее множество, а исключен только проект PROJ_ID 5. Таким образом, при использовании в своем коде этих рецептов следует внимательно учитывать задаваемые требования.

10.2. Вычисление разницы между значениями строк одной группы или сегмента

ЗАДАЧА

Требуется вернуть значения DEPTNO, ENAME и SAL для всех служащих вместе с разницей между значениями SAL для служащих одного отдела (т. е. с одинаковыми

значениями DEPTNO). Разница зарплат вычисляется между каждым текущим служащим и служащим, принятым на работу после него. Иными словами, мы хотим узнать, взаимосвязаны ли выслуга лет и зарплата на уровне отделов. Для служащих, принятых на работу последними в своих отделах, вместо разницы возвращаем значения N/A (нет данных). Результирующее множество должно выглядеть следующим образом:

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-1981	-2550
10	KING	5000	17-NOV-1981	3700
10	MILLER	1300	23-JAN-1982	N/A
20	SMITH	800	17-DEC-1980	-2175
20	JONES	2975	02-APR-1981	-25
20	FORD	3000	03-DEC-1981	0
20	SCOTT	3000	09-DEC-1982	1900
20	ADAMS	1100	12-JAN-1983	N/A
30	ALLEN	1600	20-FEB-1981	350
30	WARD	1250	22-FEB-1981	-1600
30	BLAKE	2850	01-MAY-1981	1350
30	TURNER	1500	08-SEP-1981	250
30	MARTIN	1250	28-SEP-1981	300
30	JAMES	950	03-DEC-1981	N/A

РЕШЕНИЕ

Это решение демонстрирует еще одну ситуацию, когда пригодятся оконные функции LEAD OVER и LAG OVER. Благодаря этим функциям мы можем с легкостью обращаться к предыдущим и следующим строкам без необходимости использовать дополнительные объединения. Решения с использованием альтернативных подходов, например подзапросов или самообъединений, возможны, но громоздки.

```

1 with next_sal_tab (deptno,ename,sal,hiredate,next_sal)
2 as
3 (select deptno, ename, sal, hiredate,
4     lead(sal)over(partition by deptno
5                 order by hiredate) as next_sal
6  from emp )
7
8 select deptno, ename, sal, hiredate
9     , coalesce(cast(sal-next_sal as char), 'N/A') as diff
10  from next_sal_tab

```

Разнообразия ради в этом случае мы использовали не подзапрос, а обобщенное табличное выражение. Но в настоящее время оба эти подхода будут работать с большинством СУБД, и выбор в пользу того или другого обычно делается на основе удобочитаемости.

Обсуждение

На первом шаге используем оконную функцию `LEAD OVER` для нахождения «следующей» зарплаты каждого сотрудника в своем отделе. Для служащих, принятых на работу последними в своем отделе, значение `NEXT_SAL` будет равно `NULL`:

```
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) as next_sal
from emp
```

DEPTNO	ENAME	SAL	HIREDATE	NEXT_SAL
10	CLARK	2450	09-JUN-1981	5000
10	KING	5000	17-NOV-1981	1300
10	MILLER	1300	23-JAN-1982	
20	SMITH	800	17-DEC-1980	2975
20	JONES	2975	02-APR-1981	3000
20	FORD	3000	03-DEC-1981	3000
20	SCOTT	3000	09-DEC-1982	1100
20	ADAMS	1100	12-JAN-1983	
30	ALLEN	1600	20-FEB-1981	1250
30	WARD	1250	22-FEB-1981	2850
30	BLAKE	2850	01-MAY-1981	1500
30	TURNER	1500	08-SEP-1981	1250
30	MARTIN	1250	28-SEP-1981	950
30	JAMES	950	03-DEC-1981	

На следующем шаге вычисляем разницу между зарплатой каждого служащего и зарплатой служащего, принятого на работу следующим после него в одном отделе:

```
select deptno,ename,sal,hiredate, sal-next_sal diff
from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) next_sal
from emp
)
```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-1981	-2550
10	KING	5000	17-NOV-1981	3700
10	MILLER	1300	23-JAN-1982	
20	SMITH	800	17-DEC-1980	-2175
20	JONES	2975	02-APR-1981	-25
20	FORD	3000	03-DEC-1981	0
20	SCOTT	3000	09-DEC-1982	1900
20	ADAMS	1100	12-JAN-1983	
30	ALLEN	1600	20-FEB-1981	350
30	WARD	1250	22-FEB-1981	-1600
30	BLAKE	2850	01-MAY-1981	1350

30	TURNER	1500	08-SEP-1981	250
30	MARTIN	1250	28-SEP-1981	300
30	JAMES	950	03-DEC-1981	

Затем с помощью функции `COALESCE` помечаем меткой `N/A` отсутствующие следующие зарплаты. Чтобы вернуть значение `N/A`, нужно выполнить приведение значения `DIFF` к строковому типу данных:

```
select deptno,ename,sal,hiredate,
       nvl(to_char(sal-next_sal),'N/A') diff
from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) next_sal
from emp
)
```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-1981	-2550
10	KING	5000	17-NOV-1981	3700
10	MILLER	1300	23-JAN-1982	N/A
20	SMITH	800	17-DEC-1980	-2175
20	JONES	2975	02-APR-1981	-25
20	FORD	3000	03-DEC-1981	0
20	SCOTT	3000	09-DEC-1982	1900
20	ADAMS	1100	12-JAN-1983	N/A
30	ALLEN	1600	20-FEB-1981	350
30	WARD	1250	22-FEB-1981	-1600
30	BLAKE	2850	01-MAY-1981	1350
30	TURNER	1500	08-SEP-1981	250
30	MARTIN	1250	28-SEP-1981	300
30	JAMES	950	03-DEC-1981	N/A

Хотя большинство решений в этой книге не предусматривают ситуаций «что, если» (чтобы обеспечить удобство чтения и пощадить рассудок автора), ситуация с дубликатами, которая может возникнуть при использовании функции `LEAD OVER`, все же требует рассмотрения. Данные таблицы `EMP`, поскольку она представляет собой простой пример, не содержат служащих с дубликатами значения `HIREDATE`, однако так бывает редко. Тем не менее мы обычно не рассматривали ситуацию «что, если», связанную с дубликатами значений (поскольку таблица `EMP` не содержит никаких дубликатов), однако обходное решение с использованием функций `LEAD` может быть не всем понятным и очевидным. Рассмотрим следующий запрос, который возвращает разницу между значениями `SAL` служащих отдела 10 (разница вычисляется в порядке их приема на работу):

```
select deptno,ename,sal,hiredate,
       lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
from (
select deptno,ename,sal,hiredate,
```

```

lead(sal)over(partition by deptno
              order by hiredate) next_sal
from emp
where deptno=10 and empno > 10
)

```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-1981	-2550
10	KING	5000	17-NOV-1981	3700
10	MILLER	1300	23-JAN-1982	N/A

Это решение работает должным образом с текущими данными таблицы EMP, но если бы таблица содержала дубликаты рассматриваемых значений, оно бы дало сбой. Рассмотрим следующий пример, в котором четыре других служащих были приняты на работу в тот же день, что и служащий KING:

```

insert into emp (empno,ename,deptno,sal,hiredate)
values (1,'ant',10,1000,to_date('17-NOV-1981'))

```

```

insert into emp (empno,ename,deptno,sal,hiredate)
values (2,'joe',10,1500,to_date('17-NOV-1981'))

```

```

insert into emp (empno,ename,deptno,sal,hiredate)
values (3,'jim',10,1600,to_date('17-NOV-1981'))

```

```

insert into emp (empno,ename,deptno,sal,hiredate)
values (4,'jon',10,1700,to_date('17-NOV-1981'))

```

```

select deptno,ename,sal,hiredate,
       lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno
                    order by hiredate) next_sal
from emp
where deptno=10
)

```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-1981	1450
10	ant	1000	17-NOV-1981	-500
10	joe	1500	17-NOV-1981	-3500
10	KING	5000	17-NOV-1981	3400
10	jim	1600	17-NOV-1981	-100
10	jon	1700	17-NOV-1981	400
10	MILLER	1300	23-JAN-1982	N/A

Как можно видеть, за исключением служащего JON, заработная плата всех служащих, принятых на работу в один день (17 ноября), сравнивается с заработной платой следующего служащего, принятого на работу в тот же день! Это неправильно. Зарплата всех служащих, принятых на работу 17 ноября, должна сравниваться с заработной платой служащего MILLER, а не с заработной платой другого служащего, принятого на работу 17 ноября. Возьмем, например, служащего ANT. Его значение DIFF равно -500, т. к. его зарплата сравнивается с зарплатой служащего JOE, которая на 500 больше, чем его. Правильное же значение DIFF для служащего ANT должно быть -300, поскольку его зарплата должна сравниваться с зарплатой служащего MILLER, который является следующим нанятым служащим. Решение не работает правильно с дубликатами по той причине, что функция LEAD OVER Oracle по умолчанию заглядывает только одну строку вперед. При этом для служащего ANT следующей зарплатой по HIREDATE будет зарплата служащего JOE, поскольку функция LEAD OVER смотрит только одну строку вперед и не пропускает дубликатов. К счастью, Oracle предусматривает такую ситуацию и позволяет передачу функции LEAD OVER дополнительного параметра, указывающего, насколько далеко вперед ей смотреть. В предыдущем примере для правильного решения нужно просто определить расстояние в строках между записью каждого служащего, принятого на работу 17 ноября, и записью служащего MILLER, который был принят на работу 23 января. Далее приводится соответствующий запрос и его результаты:

```
select deptno,ename,sal,hiredate,
       lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
  from (
select deptno,ename,sal,hiredate,
       lead(sal,cnt-rn+1)over(partition by deptno
                             order by hiredate) next_sal
  from (
select deptno,ename,sal,hiredate,
       count(*)over(partition by deptno,hiredate) cnt,
       row_number()over(partition by deptno,hiredate order by sal) rn
  from emp
 where deptno=10
  )
  )
```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-1981	1450
10	ant	1000	17-NOV-1981	-300
10	joe	1500	17-NOV-1981	200
10	jim	1600	17-NOV-1981	300
10	jon	1700	17-NOV-1981	400
10	KING	5000	17-NOV-1981	3700
10	MILLER	1300	23-JAN-1982	N/A

Вот теперь мы получили правильные результаты. Как можно видеть, здесь значение SAL всех служащих, нанятых на работу 17 ноября, сравниваются с зарплатой

служащего MILLER. В частности, значение DIFF для служащего ANT теперь равно -300 , каким оно и должно быть. Если с первого взгляда непонятно, что собой представляет выражение $CNT-RN+1$, так это просто расстояние в строках между записью каждого служащего, принятого на работу 17 ноября, и записью служащего MILLER. Рассмотрим следующий вложенный запрос, который возвращает значения для CNT и RN:

```
select deptno,ename,sal,hiredate,
       count(*)over(partition by deptno,hiredate) cnt,
       row_number()over(partition by deptno,hiredate order by sal) rn
from emp
where deptno=10
```

DEPTNO	ENAME	SAL	HIREDATE	CNT	RN
10	CLARK	2450	09-JUN-1981	1	1
10	ant	1000	17-NOV-1981	5	1
10	joe	1500	17-NOV-1981	5	2
10	jim	1600	17-NOV-1981	5	3
10	jon	1700	17-NOV-1981	5	4
10	KING	5000	17-NOV-1981	5	5
10	MILLER	1300	23-JAN-1982	1	1

Значение CNT показывает для каждого служащего, значения HIREDATE которого в таблице имеет дубликаты, общее количество таких дубликатов. Значения RN представляют ранг служащих отдела 10. Ранги делятся по DEPTNO и HIREDATE, поэтому значение RN больше чем 1 будут иметь только служащие с дублирующимися значениями HIREDATE. Ранги упорядочены по значению SAL — этот критерий выбран чисто произвольно, как обеспечивающий удобство ранжирования. С таким же эффектом можно было бы выбрать и столбец EMPNO. Зная общее количество дубликатов и имея ранг каждого из них, расстояние в строках до записи MILLER вычисляется просто как общее количество дубликатов минус ранг текущего дубликата плюс 1 ($CNT-RN+1$). Далее показан запрос с вычислением этого расстояния и соответствующий эффект его воздействия на результаты функции LEAD OVER:

```
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno
                    order by hiredate) incorrect,
       cnt-rn+1 distance,
       lead(sal,cnt-rn+1)over(partition by deptno
                              order by hiredate) correct
from (
select deptno,ename,sal,hiredate,
       count(*)over(partition by deptno,hiredate) cnt,
       row_number()over(partition by deptno,hiredate
                        order by sal) rn
from emp
where deptno=10
)
```

DEPTNO	ENAME	SAL	HIREDATE	INCORRECT	DISTANCE	CORRECT
10	CLARK	2450	09-JUN-1981	1000	1	1000
10	ant	1000	17-NOV-1981	1500	5	1300
10	joe	1500	17-NOV-1981	1600	4	1300
10	jim	1600	17-NOV-1981	1700	3	1300
10	jon	1700	17-NOV-1981	5000	2	1300
10	KING	5000	17-NOV-1981	1300	1	1300
10	MILLER	1300	23-JAN-1982		1	

Теперь можно ясно видеть эффект передачи правильного расстояния в строках функции `LEAD OVER`. Столбец `INCORRECT` содержит значения, возвращаемые функцией `LEAD OVER`, использующей стандартное расстояние в одну строку. А столбец `CORRECT` содержит значения, возвращаемые функцией `LEAD OVER`, использующей соответствующее расстояние в строках между служащими с дубликатами `HIREDATE` и служащим `MILLER`. Все, что осталось сделать на этом этапе, — вычислить разницу между значениями `CORRECT` и `SAL` для каждой строки. Как это сделать, было уже показано ранее.

10.3. Определение границ диапазона последовательных значений

ЗАДАЧА

Этот рецепт является расширением предыдущего и использует то же представление `V`. Определив диапазоны последовательных значений, далее мы хотим найти начальные и конечные значения этих диапазонов. В отличие от предыдущего рецепта, мы хотим возвращать даже те строки, которые не входят в набор последовательных значений. Почему? Потому, что такие строки представляют как начало, так и конец своих диапазонов. Таким образом, для данных, например, следующего представления `V`:

```
select *
  from V
```

PROJ_ID	PROJ_START	PROJ_END
1	01-JAN-2020	02-JAN-2020
2	02-JAN-2020	03-JAN-2020
3	03-JAN-2020	04-JAN-2020
4	04-JAN-2020	05-JAN-2020
5	06-JAN-2020	07-JAN-2020
6	16-JAN-2020	17-JAN-2020
7	17-JAN-2020	18-JAN-2020
8	18-JAN-2020	19-JAN-2020
9	19-JAN-2020	20-JAN-2020
10	21-JAN-2020	22-JAN-2020
11	26-JAN-2020	27-JAN-2020

```

12 27-JAN-2020 28-JAN-2020
13 28-JAN-2020 29-JAN-2020
14 29-JAN-2020 30-JAN-2020

```

мы хотим получить такое результирующее множество:

```

PROJ_GRP PROJ_START PROJ_END
-----
1 01-JAN-2020 05-JAN-2020
2 06-JAN-2020 07-JAN-2020
3 16-JAN-2020 20-JAN-2020
4 21-JAN-2020 22-JAN-2020
5 26-JAN-2020 30-JAN-2020

```

РЕШЕНИЕ

Эта задача несколько сложнее, чем предыдущая. Прежде всего нам надо идентифицировать диапазоны строк. Диапазон строк определяется значениями столбцов `PROJ_START` и `PROJ_END`. Чтобы строка считалась «последовательной», или членом группы, ее значение `PROJ_START` должно равняться значению `PROJ_END` предшествующей строки. Строка, чье значение `PROJ_START` не равно значению `PROJ_END` предшествующей строки, а также чье значение `PROJ_END` не равно значению `PROJ_START` следующей строки, является случаем группы из одной строки. Идентифицировав диапазоны строк, нужно сгруппировать строки в пределах этих диапазонов и вернуть только их начальные и конечные даты.

Рассмотрим первую строку требуемого результирующего множества. Ее значение `PROJ_START` — это значение `PROJ_START` для проекта `PROJ_ID 1` в представлении `V`, а значение `PROJ_END` — это значение `PROJ_END` для проекта `PROJ_ID 4` в представлении `V`. Поскольку после строки `PROJ_ID 4` нет никакого последовательного значения, она является последней в диапазоне последовательных значений и поэтому включается в первую группу.

Самый прямолинейный подход к решению этой задачи — использовать оконную функцию `LAG OVER`, чтобы определить, равняется ли значение `PROJ_END` каждой строки значению `PROJ_START` текущей строки, и на этом основании сгруппировать строки. Сгруппировав строки, используем агрегатные функции `MIN` и `MAX` для нахождения их начального и конечного значений:

```

1 select proj_grp, min(proj_start), max(proj_end)
2   from (
3 select proj_id,proj_start,proj_end,
4        sum(flag)over(order by proj_id) proj_grp
5   from (
6 select proj_id,proj_start,proj_end,
7        case when
8           lag(proj_end)over(order by proj_id) = proj_start
9           then 0 else 1
10        end flag

```

```

11     from V
12         ) alias1
13         ) alias2
14 group by proj_grp

```

Обсуждение

Оконная функция `LAG OVER` чрезвычайно полезна в этой ситуации, поскольку позволяет проверять значение `PROJ_END` каждой предыдущей строки без необходимости прибегать к самообъединению, использованию подзапроса или использованию представления. Далее приводится запрос с использованием функции `LAG OVER`, но без выражения `CASE`, и его результаты:

```

select proj_id,proj_start,proj_end,
       lag(proj_end)over(order by proj_id) prior_proj_end
from V

```

PROJ_ID	PROJ_START	PROJ_END	PRIOR_PROJ_END
1	01-JAN-2020	02-JAN-2020	
2	02-JAN-2020	03-JAN-2020	02-JAN-2020
3	03-JAN-2020	04-JAN-2020	03-JAN-2020
4	04-JAN-2020	05-JAN-2020	04-JAN-2020
5	06-JAN-2020	07-JAN-2020	05-JAN-2020
6	16-JAN-2020	17-JAN-2020	07-JAN-2020
7	17-JAN-2020	18-JAN-2020	17-JAN-2020
8	18-JAN-2020	19-JAN-2020	18-JAN-2020
9	19-JAN-2020	20-JAN-2020	19-JAN-2020
10	21-JAN-2020	22-JAN-2020	20-JAN-2020
11	26-JAN-2020	27-JAN-2020	22-JAN-2020
12	27-JAN-2020	28-JAN-2020	27-JAN-2020
13	28-JAN-2020	29-JAN-2020	28-JAN-2020
14	29-JAN-2020	30-JAN-2020	29-JAN-2020

Выражение `CASE` в полном решении просто сравнивает значение, возвращаемое функцией `LAG OVER`, со значением `PROJ_START` текущей строки и возвращает 0, если они совпадают, и 1 в противном случае. На следующем шаге вычисляем текущую сумму нулей и единиц, возвращенных выражением `CASE`, чтобы распределить все строки по группам. Далее приводятся соответствующий запрос и его результаты:

```

select proj_id,proj_start,proj_end,
       sum(flag)over(order by proj_id) proj_grp
from (
select proj_id,proj_start,proj_end,
       case when
         lag(proj_end)over(order by proj_id) = proj_start
         then 0 else 1
       end flag

```

```
from V
```

```
)
```

PROJ_ID	PROJ_START	PROJ_END	PROJ_GRP
1	01-JAN-2020	02-JAN-2020	1
2	02-JAN-2020	03-JAN-2020	1
3	03-JAN-2020	04-JAN-2020	1
4	04-JAN-2020	05-JAN-2020	1
5	06-JAN-2020	07-JAN-2020	2
6	16-JAN-2020	17-JAN-2020	3
7	17-JAN-2020	18-JAN-2020	3
8	18-JAN-2020	19-JAN-2020	3
9	19-JAN-2020	20-JAN-2020	3
10	21-JAN-2020	22-JAN-2020	4
11	26-JAN-2020	27-JAN-2020	5
12	27-JAN-2020	28-JAN-2020	5
13	28-JAN-2020	29-JAN-2020	5
14	29-JAN-2020	30-JAN-2020	5

Распределив все строки по группам, просто применяем агрегатные функции `MIN` и `MAX` к значениям столбцов `PROJ_START` и `PROJ_END` соответственно, чтобы сгруппировать строки по значениям, созданным в столбце текущей суммы `PROJ_GRP`.

10.4. Вставка пропущенных значений диапазона

ЗАДАЧА

Требуется вернуть количество служащих, принятых на работу в каждом году десятилетия, начиная с 1980 года, учитывая то обстоятельство, что в некоторых годах этого десятилетия никто на работу не принимался. Таким образом, нам надо получить следующее результирующее множество:

YR	CNT
1980	1
1981	10
1982	2
1983	1
1984	0
1985	0
1986	0
1987	0
1988	0
1989	0

РЕШЕНИЕ

Для решения этой задачи нам нужно возвращать 0 для лет, в которых прием служащих на работу не проводился. Но если в определенный год прием служащих не

проводился, тогда таблица EMP не содержит строк для этого года. А если этого года нет в таблице, то как тогда можно подсчитать количество принятых в нем служащих, даже если это и нулевое количество? Для реализации решения необходимо выполнить внешнее объединение. В частности, надо сгенерировать результирующее множество, содержащее все требуемые года, а затем подсчитать по таблице EMP количество служащих, принятых на работу в каждом из них.

DB2

Используя таблицу EMP в качестве сводной таблицы (т. к. она содержит 14 строк) и встроенную функцию YEAR, генерируем по одной строке для каждого года десятилетия, начинающегося с 1980 года. Затем выполняем внешнее объединение с таблицей EMP и подсчитываем количество служащих, принятых на работу в каждом году:

```

1 select x.yr, coalesce(y.cnt,0) cnt
2   from (
3 select year(min(hiredate)over()) -
4         mod(year(min(hiredate)over()),10) +
5         row_number()over()-1 yr
6   from emp fetch first 10 rows only
7     ) x
8   left join
9     (
10 select year(hiredate) yr1, count(*) cnt
11    from emp
12   group by year(hiredate)
13     ) y
14   on ( x.yr = y.yr1 )

```

Oracle

Решение для Oracle имеет такую же структуру, что и решение для DB2, но только другой синтаксис:

```

1 select x.yr, coalesce(cnt,0) cnt
2   from (
3 select extract(year from min(hiredate)over()) -
4         mod(extract(year from min(hiredate)over()),10) +
5         rownum-1 yr
6   from emp
7   where rownum <= 10
8     ) x
9   left join
10     (
11 select to_number(to_char(hiredate,'YYYY')) yr, count(*) cnt
12    from emp
13   group by to_number(to_char(hiredate,'YYYY'))
14     ) y
15   on ( x.yr = y.yr )

```

PostgreSQL и MySQL

Используя таблицу T10 в качестве сводной таблицы (т. к. она содержит 10 строк) и встроенную функцию `EXTRACT`, генерируем по одной строке для каждого года десятилетия, начинающегося с 1980 года. Затем выполняем внешнее объединение с таблицей EMP и подсчитываем количество служащих, принятых на работу в каждом году:

```

1 select y.yr, coalesce(x.cnt,0) as cnt
2   from (
3 select min_year-mod(cast(min_year as int),10)+rn as yr
4   from (
5 select (select min(extract(year from hiredate))
6         from emp) as min_year,
7         id-1 as rn
8   from t10
9        ) a
10       ) y
11  left join
12       (
13 select extract(year from hiredate) as yr, count(*) as cnt
14   from emp
15  group by extract(year from hiredate)
16         ) x
17  on ( y.yr = x.yr )

```

SQL Server

Используя таблицу EMP в качестве сводной таблицы (т. к. она содержит 14 строк) и встроенную функцию `YEAR`, генерируем по одной строке для каждого года десятилетия, начинающегося с 1980 года. Затем выполняем внешнее объединение с таблицей EMP и подсчитываем количество служащих, принятых на работу в каждом году:

```

1 select x.yr, coalesce(y.cnt,0) cnt
2   from (
3 select top (10)
4         (year(min(hiredate)over()) -
5         year(min(hiredate)over())%10)+
6         row_number()over(order by hiredate)-1 yr
7   from emp
8        ) x
9  left join
10       (
11 select year(hiredate) yr, count(*) cnt
12   from emp
13  group by year(hiredate)
14         ) y
15  on ( x.yr = y.yr )

```

Обсуждение

Несмотря на разный синтаксис решений, в них всех используется одинаковый подход. Вложенный запрос X возвращает каждый год десятилетия, начинающегося с 1980 года, сначала определяя год первого приема на работу (минимальное значение `HIREDATE`). Далее к разнице между годом первого приема на работу и остатку от деления этого года на 10 добавляется `RN-1`. Для наглядной демонстрации этого процесса просто исполните вложенный запрос X, возвращая по отдельности все участвующие значения. Далее приводится вложенный запрос X (DB2, Oracle, SQL Server), использующий оконную функцию `MIN OVER`, и результаты его исполнения:

```
select year(min(hiredate)over()) -
       mod(year(min(hiredate)over()),10) +
       row_number()over()-1 yr,
       year(min(hiredate)over()) min_year,
       mod(year(min(hiredate)over()),10) mod_yr,
       row_number()over()-1 rn
from emp fetch first 10 rows only
```

YR	MIN_YEAR	MOD_YR	RN
1980	1980	0	0
1981	1980	0	1
1982	1980	0	2
1983	1980	0	3
1984	1980	0	4
1985	1980	0	5
1986	1980	0	6
1987	1980	0	7
1988	1980	0	8
1989	1980	0	9

А здесь показан скалярный подзапрос для MySQL и PostgreSQL и результаты его исполнения:

```
select min_year-mod(min_year,10)+rn as yr,
       min_year,
       mod(min_year,10) as mod_yr
       rn
from (
select (select min(extract(year from hiredate))
       from emp) as min_year,
       id-1 as rn
from t10
) x
```

YR	MIN_YEAR	MOD_YR	RN
1980	1980	0	0
1981	1980	0	1

1982	1980	0	2
1983	1980	0	3
1984	1980	0	4
1985	1980	0	5
1986	1980	0	6
1987	1980	0	7
1988	1980	0	8
1989	1980	0	9

Вложенный запрос Y возвращает год для каждого значения HIREDATE и количество приемов на работу в этом году:

```
select year(hiredate) yr, count(*) cnt
  from emp
 group by year(hiredate)
```

YR	CNT
1980	1
1981	10
1982	2
1983	1

Наконец, выполняем внешнее объединение вложенного представления Y с вложенным представлением X, чтобы вернуть все годы, включая те, в которых не было никаких приемов на работу.

10.5. Генерирование последовательных числовых значений

ЗАДАЧА

Требуется создать «генератор строк» для использования в запросах, в которых требуется выполнить разворачивание. Например, нужно вернуть результирующее множество, аналогичное показанному далее, содержащее любое заданное количество строк:

```
ID
---
1
2
3
4
5
6
7
8
9
10
...
```

Если используемая СУБД поддерживает встроенные функции для динамического создания строк, создавать заранее сводную таблицу с фиксированным количеством строк нет надобности. В этом и состоит удобство динамического генератора строк. В противном случае для генерирования требуемых строк нужно использовать традиционную сводную таблицу с фиксированным количеством строк (которых не всегда может оказаться достаточно).

РЕШЕНИЕ

В решении показано, как вернуть десять строк, содержащих последовательные числа, начиная с 1. Но решение можно с легкостью модифицировать для возвращения любого количества строк.

Возможность создавать последовательные значения, начиная с 1, создает основу для многих других решений. Например, последовательность чисел можно добавлять к дате, чтобы создать последовательность дат. Такие последовательности чисел также можно использовать для парсинга строк.

DB2 и SQL Server

Для генерирования последовательности строк, содержащих возрастающие значения, используем рекурсивный оператор WITH. Собственно говоря, в настоящее время рекурсивное обобщенное выражение можно использовать для этой цели с большинством СУБД:

```
1 with x (id)
2 as (
3 select 1
4   union all
5 select id+1
6   from x
7   where id+1 <= 10
8 )
9 select * from x
```

Oracle

В Oracle строки можно генерировать с помощью оператора MODEL:

```
1 select array id
2   from dual
3   model
4     dimension by (0 idx)
5     measures(1 array)
6     rules iterate (10) (
7       array[iteration_number] = iteration_number+1
8     )
```

PostgreSQL

Используем функцию `GENERATE_SERIES`, специально предназначенную для генерирования строк:

```
1 select id
2   from generate_series (1, 10) x(id)
```

Обсуждение

DB2 и SQL Server

Рекурсивный оператор `WITH` инкрементирует `ID` (начиная с 1) до тех пор, пока не будет удовлетворено условие `WHERE`. Чтобы запустить процесс, нужно создать одну строку, содержащую значение 1. Это можно сделать, выбрав 1 из однострочной таблицы или, в случае с DB2, используя оператор `VALUES`, который создает результирующее множество с одной строкой.

Oracle

Решение с использованием оператора `MODEL` содержит явную команду `ITERATE`, которая позволяет создавать наборы строк. При отсутствии оператора `ITERATE` возвращается только одна строка, поскольку таблица `DUAL` содержит только одну строку:

```
select array id
   from dual
model
  dimension by (0 idx)
  measures (1 array)
  rules ( )
```

```
ID
--
1
```

Оператор `MODEL` не только позволяет обращаться к строкам как к элементам массива, но также и «создавать» или возвращать строки, отсутствующие в таблице, из которой производится выборка. В приведенном решении `IDX` — это индекс массива (номер конкретного значения в массиве), а `ARRAY` (с псевдонимом `ID`) — «массив» строк. По умолчанию первая строка содержит значение 1, а обращаться к ней можно посредством индекса `ARRAY[0]`. Отслеживать количество итераций можно с помощью встроенной функции Oracle `ITERATION_NUMBER`. В решении выполняются десять итераций, для которых функция `ITERATION_NUMBER` возвращает значения от 0 до 9. Добавляя к каждому из этих значений 1, получаем результаты от 1 до 10.

Работу оператора `MODEL` поможет визуализировать следующий запрос:

```
select 'array['||idx||'] = '||array as output
   from dual
```

```

model
  dimension by (0 idx)
  measures(1 array)
  rules iterate (10) (
    array[iteration_number] = iteration_number+1
  )

```

OUTPUT

```

-----
array[0] = 1
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5
array[5] = 6
array[6] = 7
array[7] = 8
array[8] = 9
array[9] = 10

```

PostgreSQL

Вся работа в этом решении выполняется функцией `GENERATE_SERIES`. Функция принимает три числовых параметра. Первый параметр — это начальное значение последовательности, второй — последнее значение, а третий — необязательный параметр «шаг» (приращение каждого значения). При отсутствии третьего параметра по умолчанию устанавливается шаг, равный единице.

Функция `GENERATE_SERIES` достаточно гибкая, чтобы явно не указывать в коде передаваемые в нее параметры. Предположим, например, что нам нужно вернуть пять строк со значениями от 10 до 30 с шагом 5. Соответствующее результирующее множество показано далее:

```

ID
---
10
15
20
25
30

```

Можно проявить творческий подход и создать запрос наподобие следующего:

```

select id
  from generate_series(
    (select min(deptno) from emp),
    (select max(deptno) from emp),
    5
  ) x(id)

```

Обратите внимание на то, что фактические значения, переданные в функцию `GENERATE_SERIES`, не известны при создании запроса, а создаются подзапросами при исполнении главного запроса.

10.6. Подведем итоги

Запросы для работы с диапазонами данных входят в число запросов, наиболее часто применяемых бизнес-пользователями, поскольку позволяют обработать результаты деловых операций. Однако в некоторых случаях работа с диапазонами требует определенного уровня подготовки, чтобы получать ответы. Рецепты этой главы должны помочь вам приобрести эту подготовку.

Расширенный поиск

По большому счету основное внимание этой книги до сих пор было сосредоточено на теме поиска. Мы рассмотрели разные типы запросов с использованием объединений, предикатов `WHERE` и методов группирования для поиска и возвращения требуемых данных. Но некоторые типы операций поиска выделяются из общего ряда в том смысле, что они представляют иную парадигму поиска. Например, когда требуется выводить результирующее множество постранично. Половина задачи — идентифицировать (т. е. найти) весь набор записей, которые нужно отобразить. А вторая половина состоит в последовательном поиске следующей страницы для отображения, когда пользователь просматривает записи на экране. На первый взгляд разбивка на страницы не относится к задачам поиска, но ее *можно* и рассматривать, и решать как таковую. Этот подход к решению задачи поиска и будет исследован в этой главе.

11.1. Разбивка результирующего множества на страницы

ЗАДАЧА

Требуется разбить результирующее множество на страницы, т. е. создать возможность просмотра его постранично. Например, надо вернуть из таблицы `EMP` первые пять зарплат, затем следующие пять и т. д. Наша цель — предоставить пользователю возможность просматривать по пять записей за раз, отображая следующие пять нажатием кнопки «Далее».

РЕШЕНИЕ

Поскольку в `SQL` нет такого понятия, как первая, последняя или следующая запись, нам нужно упорядочить строки, с которыми мы работаем. Только упорядочив их, можно возвращать правильные диапазоны строк.

Для упорядочивания строк используем оконную функцию `ROW_NUMBER OVER`, указывая в предикате `WHERE` окно записей, которые нужно вернуть. Например, следующий запрос возвращает записи с 1 по 5:

```
select sal
       from (
select row_number() over (order by sal) as rn,
       sal
```

```

    from emp
      ) x
  where rn between 1 and 5

```

```

SAL
----
800
950
1100
1250
1250

```

А следующий запрос возвращает записи с 6 по 10:

```

select sal
  from (
select row_number() over (order by sal) as rn,
      sal
  from emp
    ) x
  where rn between 6 and 10

```

```

SAL
-----
1300
1500
1600
2450
2850

```

Таким образом, вернуть любой диапазон строк можно, просто формируя соответствующим образом выражение `WHERE` запроса.

Обсуждение

Оконная функция `ROW_NUMBER OVER` во вложенном запросе `X` присваивает однозначный номер каждой зарплате (в возрастающем порядке, начиная с 1). Далее приводится сам вложенный запрос `X` и его результирующее множество:

```

select row_number() over (order by sal) as rn,
      sal
  from emp

```

RN	SAL
1	800
2	950
3	1100
4	1250
5	1250
6	1300

7	1500
8	1600
9	2450
10	2850
11	2975
12	3000
13	3000
14	5000

Присвоив зарплате номер, просто выбираем диапазон строк, который нужно вернуть, указав соответствующие значения `rn`.

В СУБД Oracle для формирования последовательности номеров строк можно вместо функции `ROW_NUMBER OVER` использовать функцию `ROWNUM`:

```
select sal
      from (
select sal, rownum rn
      from (
select sal
      from emp
order by sal
      )
      )
where rn between 6 and 10
```

```
SAL
-----
1300
1500
1600
2450
2850
```

Использование функции `ROWNUM` вынуждает создавать дополнительный вложенный подзапрос. Основной внутренний подзапрос сортирует строки по зарплате. Следующий подзапрос применяет к полученным строкам номера строк, и, наконец, внешний запрос `SELECT` возвращает требуемые данные.

11.2. Пропускаем *n* строк таблицы

ЗАДАЧА

Требуется выполнить выборку каждого второго служащего из таблицы `EMP`, т. е. вывести первого, третьего, пятого и т. д. Например, из следующей таблицы:

```
ENAME
-----
ADAMS
ALLEN
BLAKE
```


CLARK
FORD
JAMES
JONES
KING
MARTIN
MILLER
SCOTT
SMITH
TURNER
WARD

надо вернуть следующее результирующее множество:

```
ENAME  
-----  
ADAMS  
BLAKE  
FORD  
JONES  
MARTIN  
SCOTT  
TURNER
```

РЕШЕНИЕ

Чтобы пропустить вторую, четвертую или *n*-ю строку результирующего набора, его нужно сначала упорядочить, т. к. в противном случае понятие очередности к нему будет неприменимым.

С помощью функции `ROW_NUMBER OVER` пронумеровываем все строки, после чего можно использовать функцию деления по модулю, чтобы пропускать некоторые из них. Функция деления по модулю называется `MOD` в DB2, MySQL, PostgreSQL и Oracle. В SQL Server для выполнения этой операции используется оператор `%`. В следующем примере функция `MOD` пропускает четные строки:

```
1 select ename  
2   from (  
3 select row_number() over (order by ename) rn,  
4        ename  
5   from emp  
6        ) x  
7  where mod(rn,2) = 1
```

Обсуждение

Оконная функция `ROW_NUMBER OVER` вызывается во вложенном запросе `X` и ранжирует все строки (одинаковых рангов не допускается даже при наличии дубликатов). Далее приводятся соответствующий запрос и его результаты:

```
select row_number() over (order by ename) rn, ename
       from emp
```

```
RN ENAME
-- -----
 1 ADAMS
 2 ALLEN
 3 BLAKE
 4 CLARK
 5 FORD
 6 JAMES
 7 JONES
 8 KING
 9 MARTIN
10 MILLER
11 SCOTT
12 SMITH
13 TURNER
14 WARD
```

В завершение просто используем деление по модулю, чтобы пропустить каждую вторую строку.

11.3. Использование логики ИЛИ во внешних объединениях

ЗАДАЧА

Требуется вернуть имя и номер отдела всех служащих отделов 10 и 20, а также информацию для отделов 30 и 40, но без информации об их служащих.

Сначала мы пытаемся решить эту задачу с помощью внутреннего объединения таблиц EMP и DEPT:

```
select e.ename, d.deptno, d.dname, d.loc
       from dept d, emp e
       where d.deptno = e.deptno
          and (e.deptno = 10 or e.deptno = 20)
       order by 2
```

ENAME	DEPTNO	DNAME	LOC
CLARK	10	ACCOUNTING	NEW YORK
KING	10	ACCOUNTING	NEW YORK
MILLER	10	ACCOUNTING	NEW YORK
SMITH	20	RESEARCH	DALLAS
ADAMS	20	RESEARCH	DALLAS
FORD	20	RESEARCH	DALLAS
SCOTT	20	RESEARCH	DALLAS
JONES	20	RESEARCH	DALLAS

Но полученное результирующее множество не содержит информацию для отделов 30 и 30.

В следующей попытке мы применяем внешнее объединение этих таблиц, но все равно получаем неправильный результат:

```
select e.ename, d.deptno, d.dname, d.loc
       from dept d left join emp e
           on (d.deptno = e.deptno)
       where e.deptno = 10
           or e.deptno = 20
       order by 2
```

ENAME	DEPTNO	DNAME	LOC
CLARK	10	ACCOUNTING	NEW YORK
KING	10	ACCOUNTING	NEW YORK
MILLER	10	ACCOUNTING	NEW YORK
SMITH	20	RESEARCH	DALLAS
ADAMS	20	RESEARCH	DALLAS
FORD	20	RESEARCH	DALLAS
SCOTT	20	RESEARCH	DALLAS
JONES	20	RESEARCH	DALLAS

Правильное же результирующее множество должно быть следующим:

ENAME	DEPTNO	DNAME	LOC
CLARK	10	ACCOUNTING	NEW YORK
KING	10	ACCOUNTING	NEW YORK
MILLER	10	ACCOUNTING	NEW YORK
SMITH	20	RESEARCH	DALLAS
JONES	20	RESEARCH	DALLAS
SCOTT	20	RESEARCH	DALLAS
ADAMS	20	RESEARCH	DALLAS
FORD	20	RESEARCH	DALLAS
	30	SALES	CHICAGO
	40	OPERATIONS	BOSTON

РЕШЕНИЕ

Чтобы получить требуемое результирующее множество, условие `OR` нужно переместить в оператор `JOIN`:

```
1 select e.ename, d.deptno, d.dname, d.loc
2       from dept d left join emp e
3           on (d.deptno = e.deptno
4               and (e.deptno=10 or e.deptno=20))
5       order by 2
```

Альтернативно, можно сначала выполнить фильтрацию по `EMP.DEPTNO` во вложенном запросе, а затем выполнить внешнее объединение:

```

1 select e.ename, d.deptno, d.dname, d.loc
2     from dept d
3     left join
4         (select ename, deptno
5          from emp
6          where deptno in ( 10, 20 )
7          ) e on ( e.deptno = d.deptno )
8     order by 2

```

Обсуждение

Для СУБД DB2, MySQL, PostgreSQL и SQL Server предлагаются два решения. В первом решении условие `OR` перемещено в оператор `JOIN`, что делает его частью условия объединения. Это позволяет фильтровать строки, возвращаемые из таблицы `EMP`, не теряя информацию об отделах 30 и 40 из таблицы `DEPT`.

Во втором решении фильтрация осуществляется во вложенном запросе. В частности, вложенный запрос `E` выполняет фильтрацию по столбцу `EMP.DEPTNO` и возвращает интересующие нас строки таблицы `EMP`, над которыми затем выполняется внешнее объединение с таблицей `DEPT`. Поскольку в этом внешнем объединении таблица `DEPT` является базовой, то возвращаются все отделы, включая отделы 30 и 40.

11.4. Определение строк со взаимобратными значениями

ЗАДАЧА

В таблице, содержащей результаты двух тестов, требуется найти пары взаимобратных значений. Рассмотрим, например, следующее результирующее множество выборки из представления `V`:

```

select *
  from V

```

TEST1	TEST2
20	20
50	25
20	20
60	30
70	90
80	130
90	70
100	50
110	55
120	60
130	80
140	70

Изучив эти результаты, можно видеть, что количества баллов 70 для TEST1 и 90 для TEST2 являются взаимнообратными, т. е. таблица также содержит строку с 90 баллов для TEST1 и 70 баллов для TEST2. Аналогично, количества баллов 80 для TEST1 и 130 для TEST2 являются взаимнообратными, поскольку таблица также содержит строку с 130 баллов для TEST1 и 80 баллов для TEST2. Кроме этого, количества баллов 20 для TEST1 и 20 баллов для TEST2 являются взаимнообратными для строки, содержащей 20 баллов для TEST2 и 20 баллов для TEST1. Необходимо вывести только один набор взаимнообратных значений. Результирующее множество должно быть таким:

TEST1	TEST2
20	20
70	90
80	130

а не таким:

TEST1	TEST2
20	20
20	20
70	90
80	130
90	70
130	80

РЕШЕНИЕ

Применяем самообъединение для идентификации строк, в которых значение TEST1 равно значению TEST2 и наоборот:

```
select distinct v1.*
  from V v1, V v2
 where v1.test1 = v2.test2
       and v1.test2 = v2.test1
       and v1.test1 <= v1.test2
```

Обсуждение

Самообъединение создает декартово произведение, в котором каждое значение TEST1 можно сравнить с каждым значением TEST2, и наоборот. Далее приводится запрос для выявления строк со взаимнообратными значениями и результаты его исполнения:

```
select v1.*
  from V v1, V v2
 where v1.test1 = v2.test2
       and v1.test2 = v2.test1
```

TEST1	TEST2
20	20
20	20
20	20
20	20
90	70
130	80
70	90
80	130

Использование в решении ключевого слова `DISTINCT` обеспечивает удаление дубликатов строк из конечного результирующего множества. А последний фильтр в предикате `WHERE (and V1.TEST1 <= V1.TEST2)` обеспечит возвращение только одной пары строк с взаимнообратными значениями (в которой значение `TEST1` меньше или равно значению `TEST2`).

11.5. Выборка первых *n* записей с наибольшими значениями

ЗАДАЧА

Требуется ограничить результирующее множество определенным количеством записей на основе какого-либо ранжирования. Например, надо вернуть имена и зарплаты сотрудников с пятью наивысшими зарплатами.

РЕШЕНИЕ

Решение этой задачи основано на применении оконной функции. Какую именно оконную функцию использовать, зависит от подхода к обработке дубликатов. В следующем решении задействована функция `DENSE_RANK`, вследствие чего все дубликаты зарплат будут рассматриваться как одно значение:

```

1 select ename, sal
2   from (
3 select ename, sal,
4        dense_rank() over (order by sal desc) dr
5   from emp
6        ) x
7  where dr <= 5

```

Общее количество строк может быть больше пяти, но при этом результирующее множество будет содержать только пять однозначных значений зарплат. Чтобы вернуть пять строк, независимо от дубликатов значений зарплат, используйте функцию `ROW_NUMBER OVER` (которая не учитывает дубликаты).

Обсуждение

Вся работа решения выполняется оконной функцией `DENSE_RANK OVER` во вложенном запросе X. Далее приводится пример применения этой функции с возвратом всех строк таблицы:

```
select ename, sal,
       dense_rank() over (order by sal desc) dr
from emp
```

ENAME	SAL	DR
KING	5000	1
SCOTT	3000	2
FORD	3000	2
JONES	2975	3
BLAKE	2850	4
CLARK	2450	5
ALLEN	1600	6
TURNER	1500	7
MILLER	1300	8
WARD	1250	9
MARTIN	1250	9
ADAMS	1100	10
JAMES	950	11
SMITH	800	12

Все, что осталось теперь сделать, — это отобрать только те строки, в которых значение `DR` меньше или равно пяти.

11.6. Выявление строк с наибольшим и наименьшим значениями

ЗАДАЧА

Требуется найти строки таблицы с предельными значениями. Например, в таблице `EMP` надо найти служащих с наиболее высокой и наиболее низкой зарплатами.

РЕШЕНИЕ

Для поиска наименьшего и наибольшего значений столбца `SAL` в СУБД `DB2`, `Oracle` и `SQL Server` используем оконные функции `MIN OVER` и `MAX OVER`, соответственно:

```
1 select ename
2    from (
3 select ename, sal,
4        min(sal)over() min_sal,
5        max(sal)over() max_sal
```

```

6   from emp
7       ) x
8   where sal in (min_sal,max_sal)

```

Обсуждение

Для поиска в СУБД DB2, Oracle и SQL Server наименьшего и наибольшего значений столбца SAL используем в каждой строке оконные функции MIN OVER и MAX OVER, соответственно. Далее приводится результирующее множество, возвращаемое соответствующим вложенным запросом X:

```

select ename, sal,
       min(sal)over() min_sal,
       max(sal)over() max_sal
from emp

```

ENAME	SAL	MIN_SAL	MAX_SAL
SMITH	800	800	5000
ALLEN	1600	800	5000
WARD	1250	800	5000
JONES	2975	800	5000
MARTIN	1250	800	5000
BLAKE	2850	800	5000
CLARK	2450	800	5000
SCOTT	3000	800	5000
KING	5000	800	5000
TURNER	1500	800	5000
ADAMS	1100	800	5000
JAMES	950	800	5000
FORD	3000	800	5000
MILLER	1300	800	5000

Получив это результирующее множество, осталось лишь вернуть строки, в которых значение SAL равно MIN_SAL или MAX_SAL.

11.7. Проверка значений из следующих строк

ЗАДАЧА

Требуется найти всех служащих с меньшей зарплатой, чем у служащих, которые были приняты на работу сразу после них. Например, из следующего результирующего множества:

ENAME	SAL	HIREDATE
SMITH	800	17-DEC-80
ALLEN	1600	20-FEB-81
WARD	1250	22-FEB-81
JONES	2975	02-APR-81

BLAKE	2850	01-MAY-81
CLARK	2450	09-JUN-81
TURNER	1500	08-SEP-81
MARTIN	1250	28-SEP-81
KING	5000	17-NOV-81
JAMES	950	03-DEC-81
FORD	3000	03-DEC-81
MILLER	1300	23-JAN-82
SCOTT	3000	09-DEC-82
ADAMS	1100	12-JAN-83

наш запрос должен вернуть служащих SMITH, WARD, MARTIN, JAMES и MILLER, зарплаты которых меньше, чем у служащего, принятого на работу сразу после каждого из них.

РЕШЕНИЕ

Прежде всего нам нужно определить, что означает «следующая» строка. Чтобы строку можно было определить как следующую по отношению к текущей, нам нужно упорядочить исходное результирующее множество.

Затем для доступа к значению зарплаты следующего принятого на работу служащего можно использовать оконную функцию `LEAD OVER`. После этого осталось только сравнить две зарплаты:

```
1 select ename, sal, hiredate
2     from (
3 select ename, sal, hiredate,
4         lead(sal)over(order by hiredate) next_sal
5     from emp
6         ) alias
7 where sal < next_sal
```

Обсуждение

Оконная функция `LEAD OVER` идеально подходит для решения подобных задач. Она не только позволяет создать более удобочитаемый запрос, чем решение для других СУБД, но и предрасполагает к более гибкому решению, благодаря возможности передавать в нее аргумент, определяющий, на сколько строк вперед заглядывать (по умолчанию — на одну). Наличие возможности переходить вперед более чем на одну строку важно в случае наличия дубликатов в столбце, по которому осуществляется упорядочивание строк.

Далее приводится пример запроса, демонстрирующий легкость использования функции `LEAD OVER` для сравнения зарплаты «следующего» принятого на работу служащего:

```
select ename, sal, hiredate,
       lead(sal)over(order by hiredate) next_sal
from emp
```

ENAME	SAL	HIREDATE	NEXT_SAL
SMITH	800	17-DEC-80	1600
ALLEN	1600	20-FEB-81	1250
WARD	1250	22-FEB-81	2975
JONES	2975	02-APR-81	2850
BLAKE	2850	01-MAY-81	2450
CLARK	2450	09-JUN-81	1500
TURNER	1500	08-SEP-81	1250
MARTIN	1250	28-SEP-81	5000
KING	5000	17-NOV-81	950
JAMES	950	03-DEC-81	3000
FORD	3000	03-DEC-81	1300
MILLER	1300	23-JAN-82	3000
SCOTT	3000	09-DEC-82	1100
ADAMS	1100	12-JAN-83	

В завершение нужно вернуть только те строки, в которых значение SAL меньше значения NEXT_SAL. Функция LEAD OVER по умолчанию «заглядывает» вперед только на одну строку, и поэтому, если таблица EMP содержала бы дубликаты, в частности, нескольких служащих, принятых на работу в один день, их значения SAL сравнивались бы между собой. Такое поведение может быть как запланированным, так и нежелательным. Если целью является сравнить SAL каждого служащего с SAL следующего нанятого на работу служащего, исключая других служащих, нанятых на работу в этот же день, можно использовать следующее альтернативное решение:

```
select ename, sal, hiredate
      from (
select ename, sal, hiredate,
       lead(sal,cnt-rn+1)over(order by hiredate) next_sal
      from (
select ename,sal,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
      from emp
      )
      )
 where sal < next_sal
```

Суть решения состоит в том, чтобы вычислить количество строк между текущей строкой и строкой, с которой нужно выполнять сравнение. Например, в случае наличия пяти дубликатов первый из этих пяти нужно сравнивать со строкой, находящейся от него через пять строк. Значение CNT показывает для каждого служащего, значение HIREDATE которого в таблице имеет дубликаты, общее количество таких дубликатов. Значения RN представляют ранги служащих отдела 10. Ранги делятся по HIREDATE, поэтому значение HIREDATE больше чем 1 будут иметь только служащие с дублирующимися значениями HIREDATE. Ранги сортируются (необязательно) по столбцу EMPNO. Зная общее количество дубликатов и имея ранг каждого из них, рас-

стояние в строках до следующего `HIREDATE` вычисляется просто как общее количество дубликатов минус ранг текущего дубликата плюс 1 (`CNT-RN+1`).

См. также

Дополнительные примеры использования функции `LEAD OVER` при наличии дубликатов (и более подробное рассмотрение этого метода) приводятся в *рецептах 8.7 и 10.2*.

11.8. Смещение значений строк

ЗАДАЧА

Требуется вернуть имя и зарплату каждого служащего, а также следующие большую и меньшую зарплаты, чем этого служащего. При отсутствии более высокой или более низкой зарплаты результаты замыкаются — первое значение `SAL` указывает на последнее значение `SAL`, и наоборот. Таким образом, результирующее множество должно иметь следующий вид:

ENAME	SAL	FORWARD	REWIND
SMITH	800	950	5000
JAMES	950	1100	800
ADAMS	1100	1250	950
WARD	1250	1250	1100
MARTIN	1250	1300	1250
MILLER	1300	1500	1250
TURNER	1500	1600	1300
ALLEN	1600	2450	1500
CLARK	2450	2850	1600
BLAKE	2850	2975	2450
JONES	2975	3000	2850
SCOTT	3000	3000	2975
FORD	3000	5000	3000
KING	5000	800	3000

РЕШЕНИЕ

Наличие оконных функций `LEAD OVER` и `LAG OVER` позволяет с легкостью решить эту задачу, создавая удобочитаемые запросы. Используем эти функции для обращения к предыдущей и следующей строкам по отношению к текущей:

```

1 select ename,sal,
2       coalesce(lead(sal)over(order by sal),min(sal)over()) forward,
3       coalesce(lag(sal)over(order by sal),max(sal)over()) rewind
4 from emp

```

Обсуждение

Оконные функции `LAG OVER` и `LEAD OVER` возвращают значения из предыдущей и следующей строк, соответственно (по умолчанию, если не указаны иные параметры).

Предыдущее и следующее значения определяются в части `ORDER BY` предиката `OVER`. Как можно видеть в решении, на первом шаге возвращаются следующая и предыдущая строки относительно текущей строки (строки упорядочены по столбцу `SAL`):

```
select ename,sal,
       lead(sal)over(order by sal) forward,
       lag(sal)over(order by sal) rewind
from emp
```

ENAME	SAL	FORWARD	REWIND
SMITH	800	950	
JAMES	950	1100	800
ADAMS	1100	1250	950
WARD	1250	1250	1100
MARTIN	1250	1300	1250
MILLER	1300	1500	1250
TURNER	1500	1600	1300
ALLEN	1600	2450	1500
CLARK	2450	2850	1600
BLAKE	2850	2975	2450
JONES	2975	3000	2850
SCOTT	3000	3000	2975
FORD	3000	5000	3000
KING	5000	3000	

Обратите внимание, что значение столбца `REWIND` равно `NULL` для служащего `SMITH`, а столбца `FORWARD` равно `NULL` для служащего `KING`. Это объясняется самой низкой и самой высокой зарплатой этих двух служащих, соответственно. В требованиях задачи указывается, что при наличии значений `NULL` в столбце `FORWARD` или `REWIND` результаты должны «замыкаться». Под «замыкаться» имеется в виду, что для наивысшего значения `SAL` значение столбца `FORWARD` должно быть самым меньшим значением `SAL` таблицы `EMP`, а для самого меньшего значения `SAL` значение столбца `REWIND` должно быть наибольшим значением `SAL` в таблице `EMP`. Оконные функции `MIN OVER` и `MAX OVER`, если для них не указан сегмент или окно (т. е. скобки после оператора `OVER` пустые), возвращают самое низкое и самое высокое значение зарплаты, соответственно. Далее приводятся соответствующий запрос и его результаты:

```
select ename,sal,
       coalesce(lead(sal)over(order by sal),min(sal)over()) forward,
       coalesce(lag(sal)over(order by sal),max(sal)over()) rewind
from emp
```

ENAME	SAL	FORWARD	REWIND
SMITH	800	950	5000
JAMES	950	1100	800
ADAMS	1100	1250	950

WARD	1250	1250	1100
MARTIN	1250	1300	1250
MILLER	1300	1500	1250
TURNER	1500	1600	1300
ALLEN	1600	2450	1500
CLARK	2450	2850	1600
BLAKE	2850	2975	2450
JONES	2975	3000	2850
SCOTT	3000	3000	2975
FORD	3000	5000	3000
KING	5000	800	3000

Другое полезное свойство функций `LAG OVER` и `LEAD OVER` — возможность задавать для них количество строк для перехода вперед или назад. В примере решения рецепта мы переходим только на одну строку вперед и назад. Но перейти, например, на три строки вперед и на пять назад не представляет никаких трудностей. Нужно просто указать во втором параметре функций соответствующие значения, как показано в следующем запросе:

```
select ename,sal,
       lead(sal,3)over(order by sal) forward,
       lag(sal,5)over(order by sal) rewind
from emp
```

ENAME	SAL	FORWARD	REWIND
SMITH	800	1250	
JAMES	950	1250	
ADAMS	1100	1300	
WARD	1250	1500	
MARTIN	1250	1600	
MILLER	1300	2450	800
TURNER	1500	2850	950
ALLEN	1600	2975	1100
CLARK	2450	3000	1250
BLAKE	2850	3000	1250
JONES	2975	5000	1300
SCOTT	3000		1500
FORD	3000		1600
KING	5000		2450

11.9. Ранжирование результатов

ЗАДАЧА

Требуется выполнить ранжирование зарплат в таблице EMP с учетом дубликатов. Результирующее множество должно иметь следующий вид:

RNK	SAL
1	800
2	950
3	1100
4	1250
4	1250
5	1300
6	1500
7	1600
8	2450
9	2850
10	2975
11	3000
11	3000
12	5000

РЕШЕНИЕ

Благодаря оконным функциям создание запросов для ранжирования представляет исключительно простую задачу. Особенно полезными в этом отношении являются следующие три оконные функции: `DENSE_RANK OVER`, `ROW_NUMBER OVER` и `RANK OVER`.

Поскольку нам нужно учитывать дубликаты, используем оконную функцию `DENSE_RANK OVER`:

```
1 select dense_rank() over(order by sal) rnk, sal
2    from emp
```

Обсуждение

Вся работа решения выполняется оконной функцией `DENSE_RANK OVER`. В скобках после ключевого слова `OVER` вставляем оператор `ORDER BY`, указывающий порядок ранжирования строк. В решении используется `ORDER BY SAL`, в результате чего строки таблицы `EMP` ранжируются по зарплате (столбцу `SAL`) в возрастающем порядке.

11.10. Исключение дубликатов

ЗАДАЧА

Требуется составить список всех должностей в таблице `EMP`, при этом исключая дубликаты. Результирующее множество должно выглядеть следующим образом:

```
JOB
-----
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN
```

РЕШЕНИЕ

Все рассматриваемые здесь СУБД поддерживают ключевое слово `DISTINCT`, которое, бесспорно, является самым легким механизмом для исключения дубликатов значений из результирующего множества. Но в этом рецепте мы также рассмотрим два дополнительных способа исключения дубликатов.

Традиционный подход с использованием ключевого слова `DISTINCT`, а иногда `GROUP BY`, несомненно, работает должным образом:

- ◆ для исключения из результирующего множества дубликатов используем ключевое слово `DISTINCT`:

```
select distinct job
      from emp
```

- ◆ кроме этого, исключить дубликаты можно также с помощью оператора `GROUP BY`:

```
select job
      from emp
      group by job
```

Но с таким же успехом можно использовать и альтернативный метод, основанный на применении оконной функции `ROW_NUMBER OVER`:

```
1 select job
2     from (
3 select job,
4         row_number()over(partition by job order by job) rn
5     from emp
6         ) x
7  where rn = 1
```

Обсуждение

Это решение для СУБД DB2, Oracle и SQL Server основано на в некоторой степени нестандартном подходе к оконным функциям с сегментированием. Вследствие применения функции `PARTITION BY` в конструкции `OVER` функции `ROW_NUMBER` значение, возвращаемое функцией `ROW_NUMBER`, сбрасывается к исходному значению 1 при обнаружении каждой новой должности. Далее приводится результирующее множество, возвращаемое соответствующим вложенным запросом X:

```
select job,
       row_number()over(partition by job order by job) rn
      from emp
```

JOB	RN
ANALYST	1
ANALYST	2
CLERK	1

CLERK	2
CLERK	3
CLERK	4
MANAGER	1
MANAGER	2
MANAGER	3
PRESIDENT	1
SALESMAN	1
SALESMAN	2
SALESMAN	3
SALESMAN	4

Здесь каждой строке присваивается последовательно возрастающий номер, который сбрасывается к исходному значению 1 при каждом изменении значения JOB. Чтобы отфильтровать дубликаты, нужно просто оставить в результирующем множестве только строки, для которых значение RN равно 1.

При использовании функции ROW_NUMBER OVER обязательно должен присутствовать оператор ORDER BY (за исключением DB2), но на результат он не влияет. Какой именно из дубликатов должностей будет возвращен, не имеет значения — при условии возвращения по одному экземпляру должности.

Вернемся к рассмотрению традиционного подхода, приведенного в *разд. «Решение»* этого рецепта. В первом варианте решения демонстрируется применение ключевого слова DISTINCT для исключения из результирующего множества дубликатов. Следует иметь в виду, что это ключевое слово применяется ко всему списку SELECT — дополнительные столбцы могут изменить и в действительности изменяют результирующее множество. Рассмотрим, чем отличаются друг от друга следующие два запроса:

select distinct job	select distinct job, deptno	
from emp	from emp	
JOB	JOB	DEPTNO
-----	-----	-----
ANALYST	ANALYST	20
CLERK	CLERK	10
MANAGER	CLERK	20
PRESIDENT	CLERK	30
SALESMAN	MANAGER	10
	MANAGER	20
	MANAGER	30
	PRESIDENT	10
	SALESMAN	30

Добавляя столбец DEPTNO в список оператора SELECT, мы хотим вернуть из таблицы EMP каждую уникальную (DISTINCT) пару значений JOB/DEPTNO.

Во втором варианте решения для исключения дубликатов используется оператор GROUP BY. И хотя применение оператора GROUP BY таким образом не является редко-

стью, имейте в виду, что `GROUP BY` и `DISTINCT` представляют собой две разные и не-взаимозаменяемые конструкции. Конструкция с `GROUP BY` была включена в это решение для полноты обсуждения, т. к. вам когда-либо, несомненно, придется с ней столкнуться.

11.11. Ход конем

ЗАДАЧА

Требуется вернуть результирующее множество, содержащее имена всех служащих, их отделы, зарплаты, даты приема на работу, а также зарплату последнего принятого на работу служащего для каждого отдела. Результирующее множество должно иметь следующий вид:

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-1982	1300
10	KING	5000	17-NOV-1981	1300
10	CLARK	2450	09-JUN-1981	1300
20	ADAMS	1100	12-JAN-1982	1100
20	SCOTT	3000	09-DEC-1982	1100
20	FORD	3000	03-DEC-1981	1100
20	JONES	2975	02-APR-1981	1100
20	SMITH	800	17-DEC-1980	1100
30	JAMES	950	03-DEC-1981	950
30	MARTIN	1250	28-SEP-1981	950
30	TURNER	1500	08-SEP-1981	950
30	BLAKE	2850	01-MAY-1981	950
30	WARD	1250	22-FEB-1981	950
30	ALLEN	1600	20-FEB-1981	950

Значения в столбце `LATEST_SAL` получаются путем применения метода, называемого «ход конем», т. к. процесс их поиска похож на ход коня в шахматах. И действительно, результат определяется «ходом коня», который сначала перескакивает вдоль горизонтального ряда в нужный столбец, а затем спускается по нему на требуемую клетку. То есть, чтобы определить значение для `LATEST_SAL`, сначала нужно перейти на требуемую строку с последним значением `HIREDATE` для каждого отдела, а затем перейти в столбец `SAL` этой строки, как показано на рис. 11.1.



Термин «ход конем» придумал один смысленый сотрудник Энтони Молинаро по имени Кей Янг (Kay Young). Энтони дал ему проверить рецепты и признался, что никак не может придумать хорошее название для этого рецепта. Так как для получения значения нужно переходить в сторону на другой столбец, а затем вниз на другую строку, Кей предложил название «ход конем».

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
30	JAMES	950	03-DEC-1981	950
30	MARTIN	1250	28-SEP-1981	950
30	TURNER	1500	08-SEP-1981	950
30	BLAKE	2850	01-MAY-1981	950
30	WARD	1250	22-FEB-1981	950
30	ALLEN	1600	20-FEB-1981	950

Рис. 11.1. Значение, определяемое «ходом коня», получается перемещением сначала в сторону в требуемый столбец, а затем вниз на требуемую строку

РЕШЕНИЕ

DB2 и SQL Server

Чтобы получить значение SAL последнего принятого на работу служащего в каждом отделе, используем подзапрос с оператором CASE. Для всех других служащих возвращаем значение SAL, равное 0. А чтобы вернуть ненулевое значение SAL для отдела каждого служащего, используем оконную функцию MAX OVER во внешнем запросе:

```

1 select deptno,
2        ename,
3        sal,
4        hiredate,
5        max(latest_sal)over(partition by deptno) latest_sal
6   from (
7 select deptno,
8        ename,
9        sal,
10       hiredate,
11       case
12         when hiredate = max(hiredate)over(partition by deptno)
13          then sal else 0
14       end latest_sal
15   from emp
16  ) x
17  order by 1, 4 desc

```

Oracle

Для возвращения наибольшего значения SAL в каждом отделе (DEPTNO) используем оконную функцию MAX OVER. А чтобы получить наибольшую зарплату SAL для значе-

ния HIREDATE последнего нанятого служащего в этом отделе, используем функции DENSE_RANK и LAST, одновременно упорядочивая по HIREDATE в конструкции KEEP:

```

1 select deptno,
2        ename,
3        sal,
4        hiredate,
5        max(sal)
6        keep(dense_rank last order by hiredate)
7        over(partition by deptno) latest_sal
8   from emp
9  order by 1, 4 desc

```

Обсуждение

DB2 и SQL Server

На первом шаге применяем оконную функцию MAX OVER в выражении CASE, чтобы определить последнего нанятого служащего в каждом отделе DEPTNO. Если значение HIREDATE служащего совпадает со значением, возвращенным функцией MAX OVER, тогда с помощью выражения CASE возвращаем зарплату SAL для этого служащего, в противном случае возвращаем ноль. Далее приводятся соответствующий запрос и его результаты:

```

select deptno,
       ename,
       sal,
       hiredate,
       case
         when hiredate = max(hiredate) over(partition by deptno)
         then sal else 0
       end latest_sal
  from emp

```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	CLARK	2450	09-JUN-1981	0
10	KING	5000	17-NOV-1981	0
10	MILLER	1300	23-JAN-1982	1300
20	SMITH	800	17-DEC-1980	0
20	ADAMS	1100	12-JAN-1982	1100
20	FORD	3000	03-DEC-1981	0
20	SCOTT	3000	09-DEC-1982	0
20	JONES	2975	02-APR-1981	0
30	ALLEN	1600	20-FEB-1981	0
30	BLAKE	2850	01-MAY-1981	0
30	MARTIN	1250	28-SEP-1981	0
30	JAMES	950	03-DEC-1981	950
30	TURNER	1500	08-SEP-1981	0
30	WARD	1250	22-FEB-1981	0

Так как значение для `LATEST_SAL` будет или ноль, или зарплата `SAL` последнего принятого на работу служащего (или служащих), предыдущий запрос можно вставить во вложенный запрос и снова применить функцию `MAX OVER`, но на этот раз возвращая наибольшее ненулевое значение `LATEST_SAL` для каждого отдела:

```
select deptno,
       ename,
       sal,
       hiredate,
       max(latest_sal)over(partition by deptno) latest_sal
  from (
select deptno,
       ename,
       sal,
       hiredate,
       case
         when hiredate = max(hiredate)over(partition by deptno)
         then sal else 0
       end latest_sal
  from emp
  ) x
 order by 1, 4 desc
```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-1982	1300
10	KING	5000	17-NOV-1981	1300
10	CLARK	2450	09-JUN-1981	1300
20	ADAMS	1100	12-JAN-1982	1100
20	SCOTT	3000	09-DEC-1982	1100
20	FORD	3000	03-DEC-1981	1100
20	JONES	2975	02-APR-1981	1100
20	SMITH	800	17-DEC-1980	1100
30	JAMES	950	03-DEC-1981	950
30	MARTIN	1250	28-SEP-1981	950
30	TURNER	1500	08-SEP-1981	950
30	BLAKE	2850	01-MAY-1981	950
30	WARD	1250	22-FEB-1981	950
30	ALLEN	1600	20-FEB-1981	950

Oracle

Ключ к решению для Oracle — применение оператора `KEEP`, который позволяет ранжировать возвращенную группу или сегмент строк и работать с первой или последней строкой в группе. Посмотрим, как будет выглядеть решение без использования оператора `KEEP`:

```
select deptno,
       ename,
```

```

    sal,
    hiredate,
    max(sal) over(partition by deptno) latest_sal
from emp
order by 1, 4 desc

```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-1982	5000
10	KING	5000	17-NOV-1981	5000
10	CLARK	2450	09-JUN-1981	5000
20	ADAMS	1100	12-JAN-1982	3000
20	SCOTT	3000	09-DEC-1982	3000
20	FORD	3000	03-DEC-1981	3000
20	JONES	2975	02-APR-1981	3000
20	SMITH	800	17-DEC-1980	3000
30	JAMES	950	03-DEC-1981	2850
30	MARTIN	1250	28-SEP-1981	2850
30	TURNER	1500	08-SEP-1981	2850
30	BLAKE	2850	01-MAY-1981	2850
30	WARD	1250	22-FEB-1981	2850
30	ALLEN	1600	20-FEB-1981	2850

Без оператора `KEEP` функция `MAX OVER` вместо возвращения зарплаты `SAL` последнего принятого на работу служащего возвращает просто наибольшую зарплату для каждого отдела `DEPTNO`. Так что в этом рецепте оператор `KEEP` позволяет упорядочить зарплаты по `HIREDATE` в каждом отделе `DEPTNO`, указав для него `ORDER BY HIREDATE`. Затем функция `DENSE_RANK` ранжирует все значения `HIREDATE` в возрастающем порядке. Наконец, функция `LAST` определяет строку, к которой нужно применить агрегатную функцию — «последнюю» строку на основе ранжирования функцией `DENSE_RANK`. В нашем случае агрегатная функция `MAX` применяется к значению `SAL` строки с «последним» значением `HIREDATE`. По сути, для каждого отдела `DEPTNO` мы выбираем значение `SAL` последней по рангу даты `HIREDATE`.

Таким образом, мы ранжируем строки для каждого отдела `DEPTNO` по одному столбцу (`HIREDATE`), но затем применяем агрегатную функцию `MAX` к другому столбцу (`SAL`). Такая возможность выполнять ранжирования по одному столбцу, а агрегацию в другом — очень удобна, т. к. предоставляет нам способ избежать лишних объединений и вложенных запросов, как это делается в других решениях. Наконец, добавив оператор `OVER` после оператора `KEEP`, можно вернуть значение `SAL`, «выбранное» оператором `KEEP` в каждой строке сегмента.

Альтернативно, упорядочивание можно выполнить в нисходящем порядке по столбцу `HIREDATE` и «выбрать» первое значение `SAL`. Сравните следующие два запроса, которые возвращают одинаковое результирующее множество:

```

select deptno,
       ename,
       sal,
       hiredate,

```

```

max(sal)
  keep(dense_rank last order by hiredate)
  over(partition by deptno) latest_sal
from emp
order by 1, 4 desc

```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-1982	1300
10	KING	5000	17-NOV-1981	1300
10	CLARK	2450	09-JUN-1981	1300
20	ADAMS	1100	12-JAN-1982	1100
20	SCOTT	3000	09-DEC-1982	1100
20	FORD	3000	03-DEC-1981	1100
20	JONES	2975	02-APR-1981	1100
20	SMITH	800	17-DEC-1980	1100
30	JAMES	950	03-DEC-1981	950
30	MARTIN	1250	28-SEP-1981	950
30	TURNER	1500	08-SEP-1981	950
30	BLAKE	2850	01-MAY-1981	950
30	WARD	1250	22-FEB-1981	950
30	ALLEN	1600	20-FEB-1981	950

```

select deptno,
       ename,
       sal,
       hiredate,
       max(sal)
  keep(dense_rank first order by hiredate desc)
  over(partition by deptno) latest_sal
from emp
order by 1, 4 desc

```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-1982	1300
10	KING	5000	17-NOV-1981	1300
10	CLARK	2450	09-JUN-1981	1300
20	ADAMS	1100	12-JAN-1982	1100
20	SCOTT	3000	09-DEC-1982	1100
20	FORD	3000	03-DEC-1981	1100
20	JONES	2975	02-APR-1981	1100
20	SMITH	800	17-DEC-1980	1100
30	JAMES	950	03-DEC-1981	950
30	MARTIN	1250	28-SEP-1981	950
30	TURNER	1500	08-SEP-1981	950
30	BLAKE	2850	01-MAY-1981	950
30	WARD	1250	22-FEB-1981	950
30	ALLEN	1600	20-FEB-1981	950

11.12. Создание простых прогнозов

ЗАДАЧА

На основании текущих данных требуется вернуть дополнительные строки и столбцы, представляющие будущие действия. Рассмотрим, например, следующее результирующее множество:

```
ID ORDER_DATE  PROCESS_DATE
-----
 1 25-SEP-2005 27-SEP-2005
 2 26-SEP-2005 28-SEP-2005
 3 27-SEP-2005 29-SEP-2005
```

Нам нужно для каждой строки этого результирующего множества вернуть три строки: имеющуюся строку плюс две дополнительные строки для каждого заказа. При этом, кроме дополнительных строк, также надо вернуть два дополнительных столбца, содержащих ожидаемые даты обработки заказов.

Из предоставленного результирующего множества следует, что для обработки заказа требуются два дня. Предположим, что следующим шагом после обработки является проверка (verification), а последним — поставка (shipment). Проверка выполняется на следующий день после обработки, а поставка — на следующий день после проверки. Нам нужно вернуть результирующее множество, описывающее все эту процедуру. В конечном итоге мы хотим преобразовать исходное результирующее множество в следующее:

```
ID ORDER_DATE  PROCESS_DATE  VERIFIED  SHIPPED
-----
 1 25-SEP-2005 27-SEP-2005
 1 25-SEP-2005 27-SEP-2005 28-SEP-2005
 1 25-SEP-2005 27-SEP-2005 28-SEP-2005 29-SEP-2005
 2 26-SEP-2005 28-SEP-2005
 2 26-SEP-2005 28-SEP-2005 29-SEP-2005
 2 26-SEP-2005 28-SEP-2005 29-SEP-2005 30-SEP-2005
 3 27-SEP-2005 29-SEP-2005
 3 27-SEP-2005 29-SEP-2005 30-SEP-2005
 3 27-SEP-2005 29-SEP-2005 30-SEP-2005 01-OCT-2005
```

РЕШЕНИЕ

Ключ к решению — использовать декартово произведение для создания двух дополнительных строк в каждом заказе, а затем просто применить выражение CASE, чтобы создать требуемые значения столбцов.

DB2, MySQL и SQL Server

С помощью рекурсивного оператора WITH создаем строки, требуемые для декартова произведения. Решения для DB2 и SQL Server практически идентичны и различа-

ются только функциями для получения текущей даты: в DB2 используется функция `CURRENT_DATE`, а в SQL Server — `GETDATE`. А MySQL использует функцию `CURDATE` и требует добавления ключевого слова `RECURSIVE` после `WITH` для обозначения обобщенного табличного выражения. Далее приводится версия решения для SQL Server:

```

1 with nrows(n) as (
2 select 1 from t1 union all
3 select n+1 from nrows where n+1 <= 3
4 )
5 select id,
6     order_date,
7     process_date,
8     case when nrows.n >= 2
9         then process_date+1
10        else null
11    end as verified,
12    case when nrows.n = 3
13        then process_date+2
14        else null
15    end as shipped
16    from (
17 select nrows.n id,
18        getdate()+nrows.n as order_date,
19        getdate()+nrows.n+2 as process_date
20    from nrows
21        ) orders, nrows
22    order by 1

```

Oracle

Для создания трех строк, требуемых для декартова произведения, используем иерархический оператор `CONNECT BY`. А оператор `WITH` позволит повторно использовать результаты, возвращенные оператором `CONNECT BY`, без необходимости вызывать его снова:

```

1 with nrows as (
2 select level n
3     from dual
4 connect by level <= 3
5 )
6 select id,
7     order_date,
8     process_date,
9     case when nrows.n >= 2
10        then process_date+1
11        else null
12    end as verified,

```



```

13         case when nrows.n = 3
14             then process_date+2
15             else null
16         end as shipped
17     from (
18 select nrows.n id,
19         sysdate+nrows.n as order_date,
20         sysdate+nrows.n+2 as process_date
21     from nrows
22         ) orders, nrows

```

PostgreSQL

Декартово произведение можно создать многими разными способами — в нашем решении для этого используется функция `GENERATE_SERIES`:

```

1 select id,
2         order_date,
3         process_date,
4         case when gs.n >= 2
5             then process_date+1
6             else null
7         end as verified,
8         case when gs.n = 3
9             then process_date+2
10            else null
11        end as shipped
12     from (
13 select gs.id,
14         current_date+gs.id as order_date,
15         current_date+gs.id+2 as process_date
16     from generate_series(1,3) gs (id)
17         ) orders,
18         generate_series(1,3)gs(n)

```

MySQL

СУБД MySQL не располагает никакими функциями для автоматического создания строк.

Обсуждение

DB2, MySQL и SQL Server

Результирующее множество, представленное в *разд. «Задача»*, возвращается с помощью показанного в следующем запросе вложенного представления `ORDERS`:

```

with nrows (n) as (
select 1 from t1 union all

```

```
select n+1 from nrows where n+1 <= 3
)
select nrows.n id, getdate()+nrows.n as order_date,
       getdate()+nrows.n+2 as process_date
       from nrows
```

```
ID ORDER_DATE  PROCESS_DATE
-----
1 25-SEP-2005 27-SEP-2005
2 26-SEP-2005 28-SEP-2005
3 27-SEP-2005 29-SEP-2005
```

В этом запросе для создания трех строк, представляющих заказы для обработки, просто применяется выражение WITH. Оператор NROWS возвращает значения 1, 2 и 3, которые добавляются к GETDATE (к CURRENT_DATE — в DB2 и к CURDATE() — в MySQL) для представления дат заказов. Поскольку в *разд. «Задача»* на обработку заказов предусмотрены два дня, запрос также добавляет два дня к ORDER_DATE (добавляет к GETDATE значение, возвращенное NROWS, а затем добавляет еще два дня).

Получив базовое результирующее множество, на следующем шаге создаем декартово произведение, т. к. согласно требованиям задачи для каждого заказа необходимо вернуть три строки. Для этого используем оператор NROWS, как показано в следующем запросе:

```
with nrows (n) as (
select 1 from t1 union all
select n+1 from nrows where n+1 <= 3
)
select nrows.n,
       orders.*
       from (
select nrows.n id,
       getdate()+nrows.n as order_date,
       getdate()+nrows.n+2 as process_date
       from nrows
       ) orders, nrows
order by 2,1
```

```
N ID ORDER_DATE  PROCESS_DATE
-----
1 1 25-SEP-2005 27-SEP-2005
2 1 25-SEP-2005 27-SEP-2005
3 1 25-SEP-2005 27-SEP-2005
1 2 26-SEP-2005 28-SEP-2005
2 2 26-SEP-2005 28-SEP-2005
3 2 26-SEP-2005 28-SEP-2005
1 3 27-SEP-2005 29-SEP-2005
2 3 27-SEP-2005 29-SEP-2005
3 3 27-SEP-2005 29-SEP-2005
```

Получив три строки для каждого заказа, с помощью выражения `CASE` создаем дополнительные значения столбцов для представления состояний проверки и поставки.

Значения столбцов `VERIFIED` и `SHIPPED` для первой строки каждого заказа должны быть `NULL`. Для второй строки каждого заказа значения столбца `SHIPPED` должны быть `NULL`. А для третьей строки каждого заказа значение всех столбцов должны быть `не-NULL`. Далее показаны соответствующий запрос и конечные результаты:

```
select n+1 from nrows where n+1 <= 3
)
select id,
       order_date,
       process_date,
       case when nrows.n >= 2
            then process_date+1
            else null
       end as verified,
       case when nrows.n = 3
            then process_date+2
            else null
       end as shipped
from (
select nrows.n id,
       getdate()+nrows.n as order_date,
       getdate()+nrows.n+2 as process_date
from nrows
) orders, nrows
order by 1
```

ID	ORDER_DATE	PROCESS_DATE	VERIFIED	SHIPPED
1	25-SEP-2005	27-SEP-2005		
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	29-SEP-2005
2	26-SEP-2005	28-SEP-2005		
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	30-SEP-2005
3	27-SEP-2005	29-SEP-2005		
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	01-OCT-2005

Конечное результирующее множество выражает весь процесс обработки заказа — от дня его получения до требуемого дня поставки.

Oracle

Результирующее множество, представленное в *разд. «Задача»*, возвращается с помощью показанного в следующем запросе вложенного представления `ORDERS`:

```

with nrows as (
select level n
  from dual
connect by level <= 3
)
select nrows.n id,
       sysdate+nrows.n order_date,
       sysdate+nrows.n+2 process_date
  from nrows

```

```

ID ORDER_DATE  PROCESS_DATE
-----
1 25-SEP-2005 27-SEP-2005
2 26-SEP-2005 28-SEP-2005
3 27-SEP-2005 29-SEP-2005

```

В этом запросе для создания трех строк, представляющих заказы для обработки, просто применяется оператор `CONNECT BY`. К строкам `NROWS.N`, возвращенным оператором `CONNECT BY`, обращаемся с помощью оператора `WITH`. Оператор `CONNECT BY` возвращает значения 1, 2 и 3, которые добавляются к `SYSDATE` для представления дат заказов. Поскольку в *разд. «Задача»* на обработку заказов предусмотрены два дня, запрос также добавляет два дня к `ORDER_DATE` (добавляет к `SYSDDATE` значение, возвращенное `GETDATE_SERIES`, а затем добавляет еще два дня).

Получив базовое результирующее множество, на следующем шаге создаем декартово произведение, т. к. согласно требованиям задачи для каждого заказа необходимо вернуть три строки. Для этого используем оператор `NROWS`, как показано в следующем запросе:

```

with nrows as (
select level n
  from dual
connect by level <= 3
)
select nrows.n,
       orders.*
  from (
select nrows.n id,
       sysdate+nrows.n order_date,
       sysdate+nrows.n+2 process_date
  from nrows
  ) orders, nrows

```

```

N  ID ORDER_DATE  PROCESS_DATE
---
1  1 25-SEP-2005 27-SEP-2005
2  1 25-SEP-2005 27-SEP-2005
3  1 25-SEP-2005 27-SEP-2005
1  2 26-SEP-2005 28-SEP-2005

```

```

2  2 26-SEP-2005 28-SEP-2005
3  2 26-SEP-2005 28-SEP-2005
1  3 27-SEP-2005 29-SEP-2005
2  3 27-SEP-2005 29-SEP-2005
3  3 27-SEP-2005 29-SEP-2005

```

Получив три строки для каждого заказа, с помощью выражения `CASE` создаем дополнительные значения столбцов для представления состояний проверки и поставки.

Значения столбцов `VERIFIED` и `SHIPPED` для первой строки каждого заказа должны быть `NULL`. Для второй строки каждого заказа значения столбца `SHIPPED` должны быть `NULL`. А для третьей строки каждого заказа значение всех столбцов должны быть `не-NULL`. Далее показаны соответствующий запрос и конечные результаты:

```

with nrows as (
select level n
  from dual
 connect by level <= 3
)
select id,
       order_date,
       process_date,
       case when nrows.n >= 2
            then process_date+1
            else null
       end as verified,
       case when nrows.n = 3
            then process_date+2
            else null
       end as shipped
  from (
select nrows.n id,
       sysdate+nrows.n order_date,
       sysdate+nrows.n+2 process_date
  from nrows
  ) orders, nrows

```

ID	ORDER_DATE	PROCESS_DATE	VERIFIED	SHIPPED
1	25-SEP-2005	27-SEP-2005		
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	29-SEP-2005
2	26-SEP-2005	28-SEP-2005		
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	30-SEP-2005
3	27-SEP-2005	29-SEP-2005		
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	01-OCT-2005

Конечное результирующее множество выражает весь процесс обработки заказа — от дня его получения до требуемого дня поставки.

PostgreSQL

Результирующее множество, представленное в *разд. «Задача»*, возвращается с помощью показанного в следующем запросе вложенного представления ORDERS:

```
select gs.id,
current_date+gs.id as order_date,
current_date+gs.id+2 as process_date
  from generate_series(1,3) gs (id)
```

ID	ORDER_DATE	PROCESS_DATE
1	25-SEP-2005	27-SEP-2005
2	26-SEP-2005	28-SEP-2005
3	27-SEP-2005	29-SEP-2005

В этом запросе для создания трех строк, представляющих заказы для обработки, просто применяется функция GENERATE_SERIES, возвращающая значения 1, 2 и 3, которые добавляются к CURRENT_DATE для представления дат заказов. Поскольку в *разд. «Задача»* на обработку заказов предусмотрены два дня, запрос также добавляет два дня к ORDER_DATE (добавляет к CURRENT_DATE значение, возвращенное GETDATE_SERIES, а затем добавляет еще два дня).

Получив базовое результирующее множество, на следующем шаге создаем декартово произведение, т. к. согласно требованиям задачи для каждого заказа необходимо вернуть три строки. Для этого используем функцию GENERATE_SERIES, как показано в следующем запросе:

```
select gs.n,
       orders.*
  from (
select gs.id,
       current_date+gs.id as order_date,
       current_date+gs.id+2 as process_date
  from generate_series(1,3) gs (id)
       ) orders,
       generate_series(1,3) gs(n)
```

N	ID	ORDER_DATE	PROCESS_DATE
1	1	25-SEP-2005	27-SEP-2005
2	1	25-SEP-2005	27-SEP-2005
3	1	25-SEP-2005	27-SEP-2005
1	2	26-SEP-2005	28-SEP-2005
2	2	26-SEP-2005	28-SEP-2005
3	2	26-SEP-2005	28-SEP-2005
1	3	27-SEP-2005	29-SEP-2005
2	3	27-SEP-2005	29-SEP-2005
3	3	27-SEP-2005	29-SEP-2005

Получив три строки для каждого заказа, с помощью выражения `CASE` создаем дополнительные значения столбцов для представления состояний проверки и поставки.

Значения столбцов `VERIFIED` и `SHIPPED` для первой строки каждого заказа должны быть `NULL`. Для второй строки каждого заказа значения столбца `SHIPPED` должны быть `NULL`. А для третьей строки каждого заказа значение всех столбцов должны быть `не-NULL`. Далее показаны соответствующий запрос и конечные результаты:

```
select id,
       order_date,
       process_date,
       case when gs.n >= 2
            then process_date+1
            else null
       end as verified,
       case when gs.n = 3
            then process_date+2
            else null
       end as shipped
from (
select gs.id,
       current_date+gs.id as order_date,
       current_date+gs.id+2 as process_date
from generate_series(1,3) gs(id)
) orders,
generate_series(1,3) gs(n)
```

ID	ORDER_DATE	PROCESS_DATE	VERIFIED	SHIPPED
1	25-SEP-2005	27-SEP-2005		
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	29-SEP-2005
2	26-SEP-2005	28-SEP-2005		
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	30-SEP-2005
3	27-SEP-2005	29-SEP-2005		
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	01-OCT-2005

Конечное результирующее множество выражает весь процесс обработки заказа — от дня его получения до требуемого дня поставки.

11.13. Подведем итоги

Рецепты этой главы представляют практические задачи, которые не поддаются решению какой-либо одной функцией. В надежде на помощь в решении некоторых типов как раз таких задач бизнес-пользователи часто вынуждены обращаться к специалистам по базам данных.

Составление отчетов и форматирование результатирующих множеств

В этой главе рассматриваются запросы, которые могут быть полезными для составления отчетов. В таких запросах обычно приходится иметь дело со специфичными вопросами форматирования, а также с разными уровнями агрегации. Другой предмет рассмотрения этой главы — *транспонирование*, или разворачивание результирующих множеств, т. е. переформатирование данных с преобразованием строк в столбцы.

По большому счету эти рецепты объединяет то, что они позволяют представлять данные в другом виде или формате, нежели тот, в котором они были изначально сохранены. По мере повышения своего уровня навыков работы с транспонированием вы, несомненно, найдете ему применение и за рамками вопросов, рассматриваемых в этой главе.

12.1. Транспонирование результирующего множества в одну строку

ЗАДАЧА

Требуется преобразовать значения групп строк в столбцы по одной строке на группу. Возьмем, например, следующее результирующее множество, отображающее количество сотрудников в каждом отделе:

DEPTNO	CNT
10	3
20	5
30	6

Мы хотим переформатировать его следующим образом:

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	5	6

Это классический пример представления данных в ином виде, нежели тот, в котором они были исходно сохранены.

РЕШЕНИЕ

Для транспонирования исходного результирующего множества используем выражение CASE и агрегатную функцию SUM:

```
1 select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
2         sum(case when deptno=20 then 1 else 0 end) as deptno_20,
3         sum(case when deptno=30 then 1 else 0 end) as deptno_30
4     from emp
```

Обсуждение

Этот пример хорош тем, что представляет собой замечательное введение в предмет транспонирования. Принцип решения очень простой: к каждой строке, возвращаемой запросом, применяем выражение CASE, чтобы разложить строки на столбцы. Затем, поскольку наша задача требует подсчитать количество служащих в каждом отделе, используем агрегатную функцию SUM для подсчета всех значений DEPTNO для каждого отдела. Разобраться с тонкостями работы запроса можно, выполнив его без агрегатной функции SUM и включив в него DEPTNO для удобочитаемости:

```
select deptno,
       case when deptno=10 then 1 else 0 end as deptno_10,
       case when deptno=20 then 1 else 0 end as deptno_20,
       case when deptno=30 then 1 else 0 end as deptno_30
   from emp
  order by 1
```

DEPTNO	DEPTNO_10	DEPTNO_20	DEPTNO_30
10	1	0	0
10	1	0	0
10	1	0	0
20	0	1	0
20	0	1	0
20	0	1	0
20	0	1	0
30	0	0	1
30	0	0	1
30	0	0	1
30	0	0	1
30	0	0	1
30	0	0	1

Выражение CASE можно рассматривать как своего рода флаг, определяющий принадлежность строки к определенному отделу DEPTNO. На этом этапе преобразование строк в столбцы почти выполнено, и нужно только суммировать значения, возвращенные для DEPTNO_10, DEPTNO_20 и DEPTNO_30, а затем выполнить группирование по номеру отдела DEPTNO. Далее приводится соответствующий запрос и его результирующее множество:

```
select deptno,
       sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp
 group by deptno
```

DEPTNO	DEPTNO_10	DEPTNO_20	DEPTNO_30
10	3	0	0
20	0	5	0
30	0	0	6

По содержимому этого результирующего множества можно видеть, что оно вполне логически осмысленно. Например, в строке отдела 10 количество служащих в столбце DEPTNO_10 равно 3 и нулю во всех остальных столбцах отделов. Но так как требуется вернуть только одну строку, то на последнем шаге мы избавимся от DEPTNO и GROUP BY и просто выполним суммирование выражений CASE:

```
select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp
```

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	5	6

Далее приводится еще один подход, который иногда применяется для решения задач такого типа:

```
select max(case when deptno=10 then empcount else null end) as deptno_10,
       max(case when deptno=20 then empcount else null end) as deptno_20,
       max(case when deptno=30 then empcount else null end) as deptno_30
  from (
select deptno, count(*) as empcount
  from emp
 group by deptno
 ) x
```

Для подсчета количества служащих в каждом отделе здесь используется вложенный запрос. А выражение CASE в основном запросе преобразовывает строки в столбцы, формируя следующий результат:

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	NULL	NULL
NULL	5	NULL
NULL	NULL	6

В завершение функция MAX сворачивает столбцы в одну строку:

```
DEPTNO_10  DEPTNO_20  DEPTNO_30
-----
           3           5           6
```

12.2. Транспонирование результирующего множества в несколько строк

ЗАДАЧА

Требуется преобразовать строки в столбцы, создавая столбец для каждого значения заданного исходного столбца. Но, в отличие от предыдущего рецепта, результирующее множество должно содержать несколько строк. Подобно решениям из предыдущего рецепта, транспонирование в несколько строк является одним из фундаментальных способов реформатирования данных.

Предположим, например, что для следующей исходной таблицы надо вернуть в столбцах все должности, а в строках — служащих с этими должностями:

```
JOB          ENAME
-----
ANALYST      SCOTT
ANALYST      FORD
CLERK        SMITH
CLERK        ADAMS
CLERK        MILLER
CLERK        JAMES
MANAGER      JONES
MANAGER      CLARK
MANAGER      BLAKE
PRESIDENT    KING
SALESMAN     ALLEN
SALESMAN     MARTIN
SALESMAN     TURNER
SALESMAN     WARD
```

Конечное результирующее множество должно выглядеть следующим образом:

```
CLERKS ANALYSTS MGRS  PREZ SALES
-----
MILLER FORD      CLARK KING TURNER
JAMES  SCOTT      BLAKE      MARTIN
ADAMS                      JONES      WARD
SMITH                                ALLEN
```

РЕШЕНИЕ

В отличие от предыдущего рецепта по транспонированию, результирующее множество решения этого рецепта содержит несколько строк. В рассматриваемом слу-

чае метод из предыдущего рецепта не подойдет, т. к. для каждого значения `JOB` будет возвращаться только одно значение `ENAME` — значение `MAX(ENAME)`. Иными словами, будет возвращена только одна строка, как и в предыдущем рецепте. Чтобы решить эту проблему, каждую комбинацию `JOB/ENAME` надо сделать уникальной. Тогда при использовании агрегатной функции для удаления значений `NULL` не будут теряться никакие значения `ENAME`.

Для придания уникальности каждой комбинации `JOB/ENAME` используем функцию ранжирования `ROW_NUMBER OVER`. Полученное результирующее множество транспонируем с помощью выражения `CASE` и агрегатной функции `MAX`, при этом группируя по значению, возвращенному оконной функцией:

```

1 select max(case when job='CLERK'
2             then ename else null end) as clerks,
3        max(case when job='ANALYST'
4             then ename else null end) as analysts,
5        max(case when job='MANAGER'
6             then ename else null end) as mgrs,
7        max(case when job='PRESIDENT'
8             then ename else null end) as prez,
9        max(case when job='SALESMAN'
10            then ename else null end) as sales
11      from (
12 select job,
13        ename,
14        row_number()over(partition by job order by ename) rn
15      from emp
16     ) x
17     group by rn

```

Обсуждение

На первом шаге придаем однозначность каждой комбинации `JOB/ENAME`, используя для этого функцию ранжирования `ROW_NUMBER OVER`:

```

select job,
       ename,
       row_number()over(partition by job order by ename) rn
from emp

```

JOB	ENAME	RN
ANALYST	FORD	1
ANALYST	SCOTT	2
CLERK	ADAMS	1
CLERK	JAMES	2
CLERK	MILLER	3
CLERK	SMITH	4
MANAGER	BLAKE	1

MANAGER	CLARK	2
MANAGER	JONES	3
PRESIDENT	KING	1
SALESMAN	ALLEN	1
SALESMAN	MARTIN	2
SALESMAN	TURNER	3
SALESMAN	WARD	4

Присвоение каждому значению `ENAME` уникального «номера строки» в рамках одной должности предотвращает любые проблемы, которые в противном случае могли бы возникнуть вследствие наличия двух служащих с одинаковым именем и должностью. Наша цель — выполнить группирование по номеру строки `rn`, не допуская при этом исключения служащих из результирующего множества вследствие применения функции `max`. Этот шаг — самый важный в решении такой задачи. Если его не выполнить, то агрегация во внешнем запросе удалит требуемые строки. Рассмотрим, как будет выглядеть результирующее множество без применения функции `row_number over`, используя метод решения из предыдущего рецепта:

```
select max(case when job='CLERK'
              then ename else null end) as clerks,
       max(case when job='ANALYST'
              then ename else null end) as analysts,
       max(case when job='MANAGER'
              then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
              then ename else null end) as prez,
       max(case when job='SALESMAN'
              then ename else null end) as sales
from emp
```

CLERKS	ANALYSTS	MGRS	PREZ	SALES
-----	-----	-----	-----	-----
SMITH	SCOTT	JONES	KING	WARD

Как видим, для каждого значения `JOB` возвращается только одна строка — служащий с максимальным значением `ENAME`. При транспонировании результирующего множества функции `MIN` и `MAX` должны служить средством для удаления значений `NULL` из результирующего множества, но не ограничивать диапазон возвращаемых значений `ENAME`. Подробности этого процесса будут проясняться по ходу объяснения.

На следующем шаге используем выражение `CASE`, чтобы разместить значения `ENAME` в соответствующие столбцы должностей:

```
select rn,
       case when job='CLERK'
            then ename else null end as clerks,
       case when job='ANALYST'
            then ename else null end as analysts,
```

```

case when job='MANAGER'
    then ename else null end as mgrs,
case when job='PRESIDENT'
    then ename else null end as prez,
case when job='SALESMAN'
    then ename else null end as sales
from (
select job,
    ename,
    row_number()over(partition by job order by ename) rn
from emp
) x

```

RN	CLERKS	ANALYSTS	MGRS	PREZ	SALES
1		FORD			
2		SCOTT			
1	ADAMS				
2	JAMES				
3	MILLER				
4	SMITH				
1			BLAKE		
2			CLARK		
3			JONES		
1				KING	
1					ALLEN
2					MARTIN
3					TURNER
4					WARD

Здесь строки транспонированы в столбцы, и осталось только поудалять значения NULL, чтобы сделать результирующее множество более удобочитаемым. Для этого выполняем группирование по столбцу RN и используем агрегатную функцию MAX. (С таким же успехом можно было бы использовать функцию MIN. Функция MAX была выбрана произвольно, поскольку в каждой группе нужно агрегировать только одно значение.) Каждой комбинации RN, JOB и ENAME соответствует только одно значение. Группирование по RN в сочетании с выражениями CASE, вложенными в вызовы функции MAX, обеспечивает выбор в каждом вызове этой функции только единственного имени из группы, остальные значения которой равны NULL:

```

select max(case when job='CLERK'
    then ename else null end) as clerks,
max(case when job='ANALYST'
    then ename else null end) as analysts,
max(case when job='MANAGER'
    then ename else null end) as mgrs,
max(case when job='PRESIDENT'
    then ename else null end) as prez,

```

```

max(case when job='SALESMAN'
      then ename else null end) as sales
from (
select job,
       ename,
       row_number()over(partition by job order by ename) rn
from emp
) x
group by rn

```

```

CLERKS ANALYSTS MGRS  PREZ SALES
-----
MILLER FORD      CLARK KING TURNER
JAMES  SCOTT     BLAKE      MARTIN
ADAMS  JONES        WARD
SMITH  ALLEN

```

Подход с использованием функции ROW_NUMBER OVER для создания уникальных комбинаций строк чрезвычайно полезен для форматирования результатов запросов. Рассмотрим следующий запрос, который создает разреженный отчет, в котором служащие сгруппированы по номерам отделов и должностям:

```

select deptno dno, job,
       max(case when deptno=10
                 then ename else null end) as d10,
       max(case when deptno=20
                 then ename else null end) as d20,
       max(case when deptno=30
                 then ename else null end) as d30,
       max(case when job='CLERK'
                 then ename else null end) as clerks,
       max(case when job='ANALYST'
                 then ename else null end) as anals,
       max(case when job='MANAGER'
                 then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
                 then ename else null end) as prez,
       max(case when job='SALESMAN'
                 then ename else null end) as sales
from (
select deptno,
       job,
       ename,
       row_number()over(partition by job order by ename) rn_job,
       row_number()over(partition by job order by ename) rn_deptno
from emp
) x
group by deptno, job, rn_deptno, rn_job
order by 1

```

DNO	JOB	D10	D20	D30	CLERKS	ANALS	MGRS	PREZ	SALES
10	CLERK	MILLER			MILLER				
10	MANAGER	CLARK					CLARK		
10	PRESIDENT	KING						KING	
20	ANALYST		FORD			FORD			
20	ANALYST		SCOTT			SCOTT			
20	CLERK		ADAMS		ADAMS				
20	CLERK		SMITH		SMITH				
20	MANAGER		JONES				JONES		
30	CLERK		JAMES	JAMES					
30	MANAGER		BLAKE				BLAKE		
30	SALESMAN		ALLEN					ALLEN	
30	SALESMAN		MARTIN					MARTIN	
30	SALESMAN		TURNER					TURNER	
30	SALESMAN		WARD					WARD	

Просто откорректировав параметр, по которому выполняется группирование (отсюда и не участвующие в агрегации элементы в предыдущем списке SELECT), можно форматировать отчеты разными способами. Стоит уделить немного времени экспериментированию с изменением значений, чтобы понять, как эти форматы меняются в зависимости от содержимого конструкции GROUP BY.

12.3. Обратное транспонирование результирующего множества

ЗАДАЧА

Требуется транспонировать столбцы в строки. Например, возьмем следующее результирующее множество:

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	5	6

Надо преобразовать его таким образом, чтобы получить следующее результирующее множество:

DEPTNO	COUNTS_BY_DEPT
10	3
20	5
30	6

Возможно, некоторые из читателей заметили, что приведенное здесь исходное результирующее множество является конечным результатом первого рецепта этой главы. Чтобы сделать это множество постоянно доступным для рассматриваемого рецепта, его можно сохранить в представлении с помощью следующего запроса:


```
create view emp_cnts as
(
select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
from emp
)
```

В последующем решении и обсуждении запросы будут обращаться к этому представлению как к EMP_CNTS.

РЕШЕНИЕ

Рассмотрев требуемое результирующее множество, легко увидеть, что его можно получить, выполнив простые операции COUNT и GROUP BY над таблицей EMP. Но цель этого примера — представить, что данные хранятся не в виде строк, а, например, денормализованы и хранятся в виде нескольких столбцов.

Чтобы преобразовать столбцы в строки, используем декартово произведение. Количество столбцов, которые надо преобразовать в строки, нужно знать заранее, т. к. кардинальность табличного выражения для создания декартова произведения должна равняться по крайней мере количеству транспонируемых столбцов.

В этом решении мы не станем создавать денормализованную таблицу данных, а воспользуемся решением из первой главы и создадим «широкое» результирующее множество. Полное решение выглядит так:

```
1 select dept.deptno,
2       case dept.deptno
3         when 10 then emp_cnts.deptno_10
4         when 20 then emp_cnts.deptno_20
5         when 30 then emp_cnts.deptno_30
6       end as counts_by_dept
7   from emp_cnts cross join
8 (select deptno from dept where deptno <= 30) dept
```

Обсуждение

Таблица EMP_CNTS является денормализованным представлением, или «широким» результирующим множеством, которое требуется преобразовать в строки. Представление это имеет следующий вид:

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	5	6

Так как исходные данные организованы в три столбца, будут созданы три строки. Начнем с создания декартова произведения между вложенным представлением EMP_CNTS и каким-либо табличным выражением, содержащим как минимум три строки. Декартово произведение создается с помощью следующего кода, использующего таблицу DEPT, имеющую четыре строки:

```
select dept.deptno,
       emp_cnts.deptno_10,
       emp_cnts.deptno_20,
       emp_cnts.deptno_30
  from (
Select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp
    ) emp_cnts,
  (select deptno from dept where deptno <= 30) dept
```

DEPTNO	DEPTNO_10	DEPTNO_20	DEPTNO_30
10	3	5	6
20	3	5	6
30	3	5	6

Декартово произведение обеспечивает возвращение по одной строке для каждого столбца вложенного представления EMP_CNTS. Поскольку конечное результирующее множество должно содержать только столбец номеров отделов (DEPTNO) и количество сотрудников в каждом отделе (COUNTS_BY_DEPT), с помощью выражения CASE преобразовываем три столбца в один:

```
select dept.deptno,
       case dept.deptno
         when 10 then emp_cnts.deptno_10
         when 20 then emp_cnts.deptno_20
         when 30 then emp_cnts.deptno_30
       end as counts_by_dept
  from
    emp_cnts
 cross join (select deptno from dept where deptno <= 30) dept
```

DEPTNO	COUNTS_BY_DEPT
10	3
20	5
30	6

12.4. Обратное транспонирование результирующего множества в один столбец

ЗАДАЧА

Требуется вернуть все столбцы результирующего множества запроса в одном столбце. Например, вернуть в одном столбце значения ENAME, JOB и SAL для всех сотрудников отдела 10. Пусть значения каждого сотрудника возвращаются в трех

строках с пустой строкой между сотрудниками. Таким образом, результирующее множество должно иметь следующий вид:

```
EMPS
-----
CLARK
MANAGER
2450

KING
PRESIDENT
5000

MILLER
CLERK
1300
```

РЕШЕНИЕ

Ключ к решению — использовать рекурсивное обобщенное табличное выражение в сочетании с декартовым произведением, чтобы вернуть четыре строки для каждого сотрудника. Требуемое обобщенное табличное выражение рассматривается в *главе 10*, а в *приложении 2* предоставлена дополнительная информация. Использование декартова произведения позволит возвращать значения столбцов в отдельных строках с пустой строкой, разделяющей значения разных служащих.

Первым делом с помощью оконной функции `ROW_NUMBER OVER` ранжируем все строки по `EMPNO` (1–4). Затем с помощью выражения `CASE` преобразовываем три столбца в один (для PostgreSQL и MySQL после первого `WITH` нужно добавить ключевое слово `RECURSIVE`):

```
1 with four_rows (id)
2   as
3   (
4     select 1
5       union all
6     select id+1
7       from four_rows
8     where id < 4
9   )
10  ,
11  x_tab (ename,job,sal,rn )
12    as
13  (select e.ename,e.job,e.sal,
14         row_number()over(partition by e.empno
15         order by e.empno)
16         from emp e
17         join four_rows on 1=1
18   )
19
```

```

20 select
21   case rn
22     when 1 then ename
23     when 2 then job
24     when 3 then cast(sal as char(4))
25   end emps
26 from x_tab

```

Обсуждение

На первом шаге ранжируем всех служащих отдела 10, используя для этого оконную функцию `ROW_NUMBER OVER`:

```

select e.ename,e.job,e.sal,
       row_number()over(partition by e.empno
                        order by e.empno) rn
from emp e
where e.deptno=10

```

ENAME	JOB	SAL	RN
CLARK	MANAGER	2450	1
KING	PRESIDENT	5000	1
MILLER	CLERK	1300	1

На этом этапе ранги служащих ничего не означают. Так как сегментирование выполнялось по `EMPNO`, все три строки отдела 10 имеют ранг 1. Разные ранги появятся после добавления декартова произведения, как показано в следующих результатах:

```

with four_rows (id)
as
(select 1
 union all
 select id+1
 from four_rows
 where id < 4
 )
select e.ename,e.job,e.sal,
       row_number()over(partition by e.empno
                        order by e.empno)
from emp e
join four_rows on 1=1

```

ENAME	JOB	SAL	RN
CLARK	MANAGER	2450	1
CLARK	MANAGER	2450	2
CLARK	MANAGER	2450	3
CLARK	MANAGER	2450	4

KING	PRESIDENT	5000	1
KING	PRESIDENT	5000	2
KING	PRESIDENT	5000	3
KING	PRESIDENT	5000	4
MILLER	CLERK	1300	1
MILLER	CLERK	1300	2
MILLER	CLERK	1300	3
MILLER	CLERK	1300	4

Здесь нужно остановиться, чтобы разобраться с двумя ключевыми вопросами:

- ◆ значение `rn` для каждого служащего больше не равно 1. Сейчас это повторяющаяся последовательность значений от 1 до 4 по той причине, что оконные функции применяются после выполнения операторов `FROM` и `WHERE`. При этом сегментирование по `empno` вызывает сброс `rn` в начальное значение 1 при обнаружении записи нового служащего;
- ◆ чтобы обеспечить четыре строки для каждого служащего, было использовано рекурсивное обобщенное табличное выражение. Для SQL Server и DB2 ключевое слово `RECURSIVE` можно не применять, но для Oracle, MySQL и PostgreSQL оно требуется.

К этому моменту вся тяжелая работа выполнена и осталось только использовать выражение `CASE`, чтобы разместить значения `ENAME`, `JOB` и `SAL` всех служащих в один столбец (чтобы выражение `CASE` могло использовать значения `SAL`, их нужно привести к строковому типу с помощью оператора `CAST`):

```
with four_rows (id)
  as
  (select 1
   union all
   select id+1
   from four_rows
   where id < 4
  )
,
x_tab (ename,job,sal,rn )
  as
  (select e.ename,e.job,e.sal,
row_number()over(partition by e.empno
order by e.empno)
from emp e
join four_rows on 1=1)
  select case rn
when 1 then ename
when 2 then job
when 3 then cast(sal as char(4))
end emps
from x_tab
```

```
EMPS
-----
CLARK
MANAGER
2450

KING
PRESIDENT
5000

MILLER
CLERK
1300
```

12.5. Исключение повторяющихся значений из результирующего множества

ЗАДАЧА

При генерировании отчета повторяющиеся одинаковые значения столбца требуется отображать только один раз. Например, при возвращении столбцов DEPTNO и ENAME таблицы надо сгруппировать все строки для каждого отдела, при этом отображая каждое значение DEPTNO только один раз. Таким образом, результирующее множество должно иметь следующий вид:

```
DEPTNO ENAME
-----
 10 CLARK
    KING
    MILLER
 20 SMITH
    ADAMS
    FORD
    SCOTT
    JONES
 30 ALLEN
    BLAKE
    MARTIN
    JAMES
    TURNER
    WARD
```

РЕШЕНИЕ

Эта простая задача на форматирование легко решается с помощью оконной функции LAG OVER:

```

1 select
2     case when
3         lag(deptno)over(order by deptno) = deptno then null
4         else deptno end DEPTNO
5     ,   ename
6 from emp

```

Для СУБД Oracle в качестве альтернативы выражению CASE можно использовать функцию DECODE:

```

1 select to_number(
2     decode(lag(deptno)over(order by deptno),
3         deptno,null,deptno)
4     ) deptno, ename
5 from emp

```

Обсуждение

На первом шаге возвращаем предшествующее значение DEPTNO для каждой строки, используя для этого оконную функцию LAG OVER:

```

select lag(deptno)over(order by deptno) lag_deptno,
       deptno,
       ename
from emp

```

LAG_DEPTNO	DEPTNO	ENAME
	10	CLARK
10	10	KING
10	10	MILLER
10	20	SMITH
20	20	ADAMS
20	20	FORD
20	20	SCOTT
20	20	JONES
20	30	ALLEN
30	30	BLAKE
30	30	MARTIN
30	30	JAMES
30	30	TURNER
30	30	WARD

В возвращенном результирующем множестве легко увидеть, где значение DEPTNO совпадает со значением LAG_DEPTNO. Для этих строк DEPTNO нужно присвоить значение NULL с помощью функции DECODE (функция TO_NUMBER включена, чтобы привести значение DEPTNO к числовому типу):

```

select to_number(
       CASE WHEN (lag(deptno)over(order by deptno)

```

```
= deptno THEN null else deptno END deptno ,
      deptno,null,deptno)
  ) deptno, ename
  from emp
```

```
DEPTNO ENAME
-----
```

```
10 CLARK
   KING
   MILLER
20 SMITH
   ADAMS
   FORD
   SCOTT
   JONES
30 ALLEN
   BLAKE
   MARTIN
   JAMES
   TURNER
   WARD
```

12.6. Транспонирование результирующего множества для облегчения вычислений с несколькими строками

ЗАДАЧА

Требуется выполнить вычисления над данными, представленными в нескольких строках. Чтобы облегчить задачу, мы хотим развернуть эти строки в столбцы, чтобы все требуемые значения были в одной строке.

В используемых для примера данных DEPTNO 20 — отдел с наибольшей совокупной зарплатой, в чем можно убедиться с помощью следующего запроса:

```
select deptno, sum(sal) as sal
  from emp
 group by deptno
```

```
DEPTNO      SAL
-----
10          8750
20         10875
30          9400
```

Нам нужно вычислить разницу между совокупными зарплатами отделов 20 и 10 и отделов 20 и 30.

Конечное результирующее множество должно выглядеть следующим образом:

```
  d20_10_diff d20_30_diff
-----
2125         1475
```


РЕШЕНИЕ

Сначала транспонируем значения общих зарплат с помощью агрегатной функции SUM и выражений CASE, а затем включаем соответствующие выражения в список SELECT:

```

1 select d20_sal - d10_sal as d20_10_diff,
2         d20_sal - d30_sal as d20_30_diff
3   from (
4 select sum(case when deptno=10 then sal end) as d10_sal,
5         sum(case when deptno=20 then sal end) as d20_sal,
6         sum(case when deptno=30 then sal end) as d30_sal
7   from emp
8        ) totals_by_dept

```

Эту задачу также можно решить с помощью запроса с обобщенным табличным выражением, который может быть более удобным для понимания:

```

with totals_by_dept (d10_sal, d20_sal, d30_sal)
as
(select
    sum(case when deptno=10 then sal end) as d10_sal,
    sum(case when deptno=20 then sal end) as d20_sal,
    sum(case when deptno=30 then sal end) as d30_sal
 from emp)
select d20_sal - d10_sal as d20_10_diff,
       d20_sal - d30_sal as d20_30_diff
 from totals_by_dept

```

Обсуждение

На первом шаге транспонируем строки с зарплатами каждого отдела в столбцы, используя для этого выражение CASE:

```

select case when deptno=10 then sal end as d10_sal,
       case when deptno=20 then sal end as d20_sal,
       case when deptno=30 then sal end as d30_sal
 from emp

```

D10_SAL	D20_SAL	D30_SAL
	800	1600
		1250
	2975	1250
		2850
2450		
	3000	
5000		
		1500
	1100	

```

          950
    3000
1300

```

Затем суммируем все зарплаты для каждого отдела, применяя агрегатную функцию SUM к каждому выражению CASE:

```

select sum(case when deptno=10 then sal end) as d10_sal,
       sum(case when deptno=20 then sal end) as d20_sal,
       sum(case when deptno=30 then sal end) as d30_sal
from emp

```

D10_SAL	D20_SAL	D30_SAL
8750	10875	9400

В завершение просто вставляем указанный код SQL во вложенный запрос и вычисляем разности.

12.7. Создание блоков данных фиксированного размера

ЗАДАЧА

Требуется организовать данные в блоки одинакового размера с предопределенным количеством элементов в каждом блоке. Общее количество блоков может быть неизвестно, но каждый блок должен содержать пять элементов. Например, надо сгруппировать по значению EMPNO служащих, содержащихся в таблице EMP, в блоки по пять служащих в каждом, как показано в следующей таблице:

GRP	EMPNO	ENAME
1	7369	SMITH
1	7499	ALLEN
1	7521	WARD
1	7566	JONES
1	7654	MARTIN
2	7698	BLAKE
2	7782	CLARK
2	7788	SCOTT
2	7839	KING
2	7844	TURNER
3	7876	ADAMS
3	7900	JAMES
3	7902	FORD
3	7934	MILLER

РЕШЕНИЕ

Решение этой задачи значительно упрощается из-за наличия в СУБД функций для ранжирования строк. После ранжирования строк блоки по пять служащих в каждом

создаются простой операцией деления с последующим округлением частного в верхнюю сторону.

Для ранжирования служащих по их EMPNO используем оконную функцию ROW_NUMBER OVER. Затем, чтобы создать группы, делим полученные ранги на 5, округляя частное в верхнюю сторону с помощью функции CEIL (для SQL Server вместо CEIL используется функция CEILING):

```
1 select ceil(row_number()over(order by empno)/5.0) grp,
2         empno,
3         ename
4         from emp
```

Обсуждение

На первом шаге оконная функция ROW_NUMBER OVER присваивает ранг, или «номер строки», всем строкам, отсортированным по значению EMPNO:

```
select row_number()over(order by empno) rn,
empno,
ename
from emp
```

RN	EMPNO	ENAME
1	7369	SMITH
2	7499	ALLEN
3	7521	WARD
4	7566	JONES
5	7654	MARTIN
6	7698	BLAKE
7	7782	CLARK
8	7788	SCOTT
9	7839	KING
10	7844	TURNER
11	7876	ADAMS
12	7900	JAMES
13	7902	FORD
14	7934	MILLER

Затем делим значения, возвращенные ROW_NUMBER OVER, на пять и применяем к полученному частному функцию CEIL (или CEILING – для SQL Server). Деление на 5 логически организует строки в группы по пять. В частности, пять значений, меньших или равных 1, пять значений больших чем 1, но меньших или равных 2 и четыре значения больших чем 2, но меньших или равных 3. Количество элементов в последней группе обусловлено не кратным пяти количеством строк в таблице EMP — 14.

Функция CEIL возвращает наименьшее целое число, большее, чем переданное ей значение (т. е. округляет его в большую сторону). Таким образом создаются группы

с целочисленными значениями. Далее приводятся соответствующий запрос с делением и применением функции `CEIL` и его результаты. Порядок выполнения операций можно проследить слева направо — от `RN` до `DIVISION` и `GRP`:

```
select row_number() over (order by empno) rn,
       row_number() over (order by empno)/5.0 division,
       ceil(row_number() over (order by empno)/5.0) grp,
       empno,
       ename
from emp
```

RN	DIVISION	GRP	EMPNO	ENAME
1	.2	1	7369	SMITH
2	.4	1	7499	ALLEN
3	.6	1	7521	WARD
4	.8	1	7566	JONES
5	1	1	7654	MARTIN
6	1.2	2	7698	BLAKE
7	1.4	2	7782	CLARK
8	1.6	2	7788	SCOTT
9	1.8	2	7839	KING
10	2	2	7844	TURNER
11	2.2	3	7876	ADAMS
12	2.4	3	7900	JAMES
13	2.6	3	7902	FORD
14	2.8	3	7934	MILLER

12.8. Создание предопределенного количества блоков данных

ЗАДАЧА

Требуется организовать данные в фиксированное количество блоков. Например, сгруппировать служащих, содержащихся в таблице `EMP`, в четыре блока. Конечное результирующее множество должно выглядеть подобным этому:

GRP	EMPNO	ENAME
1	7369	SMITH
1	7499	ALLEN
1	7521	WARD
1	7566	JONES
2	7654	MARTIN
2	7698	BLAKE
2	7782	CLARK
2	7788	SCOTT

```
3 7839 KING
3 7844 TURNER
3 7876 ADAMS
4 7900 JAMES
4 7902 FORD
4 7934 MILLER
```

Это распространенный способ организации категорийных данных, т. к. разбиение набора данных на несколько меньших наборов одинакового размера является важным шагом в разнообразных исследованиях. Например, вычисление для таких групп средней зарплаты или другого параметра может выявить тенденцию, скрываемую вариациями данных, рассматриваемых индивидуально.

Рассматриваемая здесь задача противоположна задаче из предыдущего рецепта с неизвестным количеством блоков, но заданным количеством элементов в каждом блоке. Цель этого рецепта состоит в том, чтобы создать определенное известное количество блоков данных, при этом не обязательно зная, сколько элементов будет в каждом из этих блоков.

РЕШЕНИЕ

Решение этой задачи значительно упрощается широкой доступностью функции `NTILE`, которая делит упорядоченное множество на заданное количество блоков, при этом любые элементы остатка распределяются в доступные блоки, начиная с первого. Указанное обстоятельство отражается в конечном результирующем множестве решения этого рецепта: блоки 1 и 2 содержат по четыре строки, а блоки 3 и 4 — по три.

Запрос с использованием функции `NTILE` для создания четырех блоков выглядит следующим образом:

```
1 select ntile(4)over(order by empno) grp,
2        empno,
3        ename
4    from emp
```

Обсуждение

Все работа в этом решении выполняется функцией `NTILE`. Оператор `ORDER BY` упорядочивает строки требуемым образом, после чего функция присваивает каждой строке номер группы, чтобы, например, в нашем случае первая четверть строк поместилась в первую группу, вторая четверть — во вторую и т. д.

12.9. Создание горизонтальных гистограмм

ЗАДАЧА

Требуется с помощью кода SQL создать горизонтальные гистограммы. Например, надо отобразить количество служащих в каждом отделе в виде горизонтальной гис-

тограммы, где каждый служащий представлен отдельным символом *. Таким образом, результирующее множество должно иметь следующий вид:

```
DEPTNO CNT
-----
10 ***
20 *****
30 *****
```

РЕШЕНИЕ

Ключ к решению этой задачи — использовать агрегатную функцию `COUNT` совместно с конструкцией `GROUP BY DEPTNO`, чтобы определить количество служащих в каждом отделе. Затем значение, возвращенное функцией `COUNT`, передается строковой функции, которая генерирует последовательности символов *.

DB2

Для создания гистограммы используем функцию `REPEAT`:

```
1 select deptno,
2     repeat('*',count(*)) cnt
3   from emp
4  group by deptno
```

Oracle, PostgreSQL и MySQL

Для создания последовательностей символов * требуемой длины используем функцию `LPAD`:

```
1 select deptno,
2     lpad('*',count(*),'*') as cnt
3   from emp
4  group by deptno
```

SQL Server

Для создания гистограммы используем функцию `REPEAT`:

```
1 select deptno,
2     replicate('*',count(*)) cnt
3   from emp
4  group by deptno
```

Обсуждение

В решениях для всех СУБД используется один и тот же подход — различие только в строковой функции, используемой для возвращения символа * для каждого служащего. В этом обсуждении мы воспользуемся решением для Oracle, но приводимые объяснения применимы ко всем решениям.

На первом шаге подсчитываем количество служащих в каждом отделе:

```
select deptno,
       count(*)
  from emp
 group by deptno
```

```
DEPTNO  COUNT(*)
-----  -
10      3
20      5
30      6
```

Затем используем возвращенные функцией `COUNT` значения для управления количеством символов `*` для каждого отдела. Для этого просто передаем `COUNT(*)` в качестве аргумента строковой функции `LPAD`:

```
select deptno,
       lpad('*',count(*),'*') as cnt
  from emp
 group by deptno
```

```
DEPTNO CNT
-----
10 ***
20 *****
30 *****
```

Для СУБД PostgreSQL нужно выполнять явное приведение возвращаемого функцией `COUNT(*)` значения к целочисленному типу:

```
select deptno,
       lpad('*',count(*)::integer,'*') as cnt
  from emp
 group by deptno
```

```
DEPTNO CNT
-----
10 ***
20 *****
30 *****
```

Такое приведение необходимо, поскольку в PostgreSQL числовой аргумент, передаваемый функции `LPAD`, должен быть целочисленного типа.

12.10. Создание вертикальных гистограмм

ЗАДАЧА

Требуется создать вертикальную гистограмму, растущую снизу вверх. Например, надо отобразить количество служащих в каждом отделе в виде вертикальной гистограммы, где каждый служащий представлен отдельным символом `*`. Таким образом, результирующее множество должно иметь следующий вид:

```

D10 D20 D30
--- --- ---
      *
     * *
    * *
   * * *
  * * *
 * * *

```

РЕШЕНИЕ

Способ для решения этой задачи основан на методе, уже рассмотренном в этой главе ранее, — использование функции `ROW_NUMBER OVER` для уникальной идентификации каждого экземпляра символа `*` для каждого значения `DEPTNO`. Чтобы транспонировать результирующее множество и выполнить группирование значений, возвращенных функцией `ROW_NUMBER OVER`, используем агрегатную функцию `MAX` (для SQL Server в операторе `ORDER BY` параметр `DESC` использоваться не должен):

```

1 select max(deptno_10) d10,
2         max(deptno_20) d20,
3         max(deptno_30) d30
4   from (
5 select row_number()over(partition by deptno order by empno) rn,
6        case when deptno=10 then '*' else null end deptno_10,
7        case when deptno=20 then '*' else null end deptno_20,
8        case when deptno=30 then '*' else null end deptno_30
9   from emp
10  ) x
11  group by rn
12  order by 1 desc, 2 desc, 3 desc

```

Обсуждение

На первом шаге уникально идентифицируем каждый экземпляр `*` в каждом отделе, используя для этого оконную функцию `ROW_NUMBER OVER`. И с помощью выражения `CASE` возвращаем символ `*` для каждого служащего каждого отдела:

```

select row_number()over(partition by deptno order by empno) rn,
       case when deptno=10 then '*' else null end deptno_10,
       case when deptno=20 then '*' else null end deptno_20,
       case when deptno=30 then '*' else null end deptno_30
from emp

```

```

RN  DEPTNO_10  DEPTNO_20  DEPTNO_30
---  -
1  *
2  *
3  *
1  *
2  *

```



```

3          *
4          *
5          *
1          *
2          *
3          *
4          *
5          *
6          *

```

На следующем — и последнем — шаге применяем агрегатную функцию `MAX` с каждым выражением `CASE`, группируя строки по значениям `RN`, чтобы удалить значения `NULL` из результирующего множества. И упорядочиваем результаты в возрастающем (`ASC`) или убывающем (`DESC`) порядке, в зависимости от того, как используемая СУБД сортирует значения `NULL`:

```

select max(deptno_10) d10,
       max(deptno_20) d20,
       max(deptno_30) d30
  from (
select row_number() over (partition by deptno order by empno) rn,
       case when deptno=10 then '*' else null end deptno_10,
       case when deptno=20 then '*' else null end deptno_20,
       case when deptno=30 then '*' else null end deptno_30
  from emp
   ) x
 group by rn
 order by 1 desc, 2 desc, 3 desc

```

```

D10 D20 D30
--- --- ---
*
* *
* *
* * *
* * *
* * *

```

12.11. Возвращение столбцов, не указанных в операторе *GROUP BY*

ЗАДАЧА

При исполнении запроса с оператором `GROUP BY` требуется вернуть столбцы, указанные в списке `SELECT`, но не указанные в операторе `GROUP BY`. Обычно это невозможно, поскольку такие несгруппированные столбцы представляют собой несколько решений, тогда как требуется одно значение на строку.

Предположим, например, что нам нужно определить служащих с наименьшей и наибольшей заработной платой в каждом отделе, а также в каждой должности. Для каждого служащего надо вернуть его имя, отдел, должность и зарплату. Таким образом, результирующее множество должно иметь следующий вид:

DEPTNO	ENAME	JOB	SAL	DEPT_STATUS	JOB_STATUS
10	MILLER	CLERK	1300	LOW SAL IN DEPT	TOP SAL IN JOB
10	CLARK	MANAGER	2450		LOW SAL IN JOB
10	KING	PRESIDENT	5000	TOP SAL IN DEPT	TOP SAL IN JOB
20	SCOTT	ANALYST	3000	TOP SAL IN DEPT	TOP SAL IN JOB
20	FORD	ANALYST	3000	TOP SAL IN DEPT	TOP SAL IN JOB
20	SMITH	CLERK	800	LOW SAL IN DEPT	LOW SAL IN JOB
20	JONES	MANAGER	2975		TOP SAL IN JOB
30	JAMES	CLERK	950	LOW SAL IN DEPT	
30	MARTIN	SALESMAN	1250		LOW SAL IN JOB
30	WARD	SALESMAN	1250		LOW SAL IN JOB
30	ALLEN	SALESMAN	1600		TOP SAL IN JOB
30	BLAKE	MANAGER	2850	TOP SAL IN DEPT	

К сожалению, включение всех этих столбцов в список оператора `SELECT` нарушит группирование. Рассмотрим следующий пример: служащий `KING` имеет наибольшую зарплату. Нам нужно подтвердить это следующим запросом:

```
select ename,max(sal)
      from emp
      group by ename
```

Увы, вместо ожидаемого результата в виде имени `KING` и соответствующей зарплаты этот запрос возвратит все 14 строк таблицы `EMP`. Причиной этого является группирование: функция `MAX(SAL)` применяется к каждому значению `ENAME`. Таким образом, когда по идее приведенный запрос должен означать «найти служащего с наибольшей зарплатой», в действительности он означает «найти наибольшую зарплату для каждого значения `ENAME`». В этом рецепте рассматривается метод включения в результирующее множество столбца `ENAME` без необходимости указывать его в списке оператора `GROUP BY`.

РЕШЕНИЕ

Предельные значения зарплат по отделам (`DEPTNO`) и должностям (`JOB`) находим с помощью вложенного представления. Затем выбираем только служащих с такими зарплатами.

С помощью оконных функций `MAX OVER` и `MIN OVER` находим наибольшую и наименьшую зарплату по `DEPTNO` и `JOB`. Затем выбираем строки с зарплатами, соответствующими полученным наибольшим и наименьшим значениям:

```
1 select deptno,ename,job,sal,
2       case when sal = max_by_dept
3       then 'TOP SAL IN DEPT'
```

```

4         when sal = min_by_dept
5         then 'LOW SAL IN DEPT'
6     end dept_status,
7     case when sal = max_by_job
8         then 'TOP SAL IN JOB'
9         when sal = min_by_job
10        then 'LOW SAL IN JOB'
11    end job_status
12 from (
13 select deptno,ename,job,sal,
14        max(sal)over(partition by deptno) max_by_dept,
15        max(sal)over(partition by job) max_by_job,
16        min(sal)over(partition by deptno) min_by_dept,
17        min(sal)over(partition by job) min_by_job
18    from emp
19        ) emp_sals
20 where sal in (max_by_dept,max_by_job,
21             min_by_dept,min_by_job)

```

Обсуждение

На первом шаге с помощью оконных функций `MAX OVER` и `MIN OVER` находим наибольшую и наименьшую зарплаты по `DEPTNO` и `JOB`:

```

select deptno,ename,job,sal,
       max(sal)over(partition by deptno) maxDEPT,
       max(sal)over(partition by job) maxJOB,
       min(sal)over(partition by deptno) minDEPT,
       min(sal)over(partition by job) minJOB
from emp

```

DEPTNO	ENAME	JOB	SAL	MAXDEPT	MAXJOB	MINDEPT	MINJOB
10	MILLER	CLERK	1300	5000	1300	1300	800
10	CLARK	MANAGER	2450	5000	2975	1300	2450
10	KING	PRESIDENT	5000	5000	5000	1300	5000
20	SCOTT	ANALYST	3000	3000	3000	800	3000
20	FORD	ANALYST	3000	3000	3000	800	3000
20	SMITH	CLERK	800	3000	1300	800	800
20	JONES	MANAGER	2975	3000	2975	800	2450
20	ADAMS	CLERK	1100	3000	1300	800	800
30	JAMES	CLERK	950	2850	1300	950	800
30	MARTIN	SALESMAN	1250	2850	1600	950	1250
30	TURNER	SALESMAN	1500	2850	1600	950	1250
30	WARD	SALESMAN	1250	2850	1600	950	1250
30	ALLEN	SALESMAN	1600	2850	1600	950	1250
30	BLAKE	MANAGER	2850	2850	2975	950	2450

Здесь все зарплаты сравниваются с наибольшей и наименьшей зарплатами по DEPTNO и JOB. Обратите внимание, что группирование (включение нескольких столбцов в конструкции SELECT) не затрагивает значений, возвращаемых функциями MIN OVER и MAX OVER. В этом и прелесть оконных функций: общее вычисляется для заданной «группы» или сегмента, возвращая несколько строк для каждой группы. В завершение просто вставляем оконные функции во вложенный запрос и возвращаем только те строки, которые соответствуют значениям, возвращенным оконными функциями. Для отображения «статуса» каждого служащего в конечных результатах используем выражение CASE:

```
select deptno,ename,job,sal,
       case when sal = max_by_dept
            then 'TOP SAL IN DEPT'
            when sal = min_by_dept
            then 'LOW SAL IN DEPT'
       end dept_status,
       case when sal = max_by_job
            then 'TOP SAL IN JOB'
            when sal = min_by_job
            then 'LOW SAL IN JOB'
       end job_status
from (
select deptno,ename,job,sal,
       max(sal)over(partition by deptno) max_by_dept,
       max(sal)over(partition by job) max_by_job,
       min(sal)over(partition by deptno) min_by_dept,
       min(sal)over(partition by job) min_by_job
from emp
) x
where sal in (max_by_dept,max_by_job,
            min_by_dept,min_by_job)
```

DEPTNO	ENAME	JOB	SAL	DEPT_STATUS	JOB_STATUS
10	MILLER	CLERK	1300	LOW SAL IN DEPT	TOP SAL IN JOB
10	CLARK	MANAGER	2450		LOW SAL IN JOB
10	KING	PRESIDENT	5000	TOP SAL IN DEPT	TOP SAL IN JOB
20	SCOTT	ANALYST	3000	TOP SAL IN DEPT	TOP SAL IN JOB
20	FORD	ANALYST	3000	TOP SAL IN DEPT	TOP SAL IN JOB
20	SMITH	CLERK	800	LOW SAL IN DEPT	LOW SAL IN JOB
20	JONES	MANAGER	2975		TOP SAL IN JOB
30	JAMES	CLERK	950	LOW SAL IN DEPT	
30	MARTIN	SALESMAN	1250		LOW SAL IN JOB
30	WARD	SALESMAN	1250		LOW SAL IN JOB
30	ALLEN	SALESMAN	1600		TOP SAL IN JOB
30	BLAKE	MANAGER	2850	TOP SAL IN DEPT	

12.12. Вычисление простых подсумм

ЗАДАЧА

В этом рецепте примем, что *простая подсумма* — это результирующее множество, содержащее значения агрегаций одного столбца, а также общую сумму таблицы. Для примера возьмем результирующее множество, суммирующее зарплаты в таблице EMP по должностям JOB, а также общую сумму все зарплат в таблице EMP. Общие зарплаты по должностям — это подсуммы, а сумма всех зарплат в таблице EMP — это общая сумма. Соответствующее результирующее множество должно выглядеть следующим образом:

JOB	SAL
-----	-----
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
TOTAL	29025

РЕШЕНИЕ

Для решения этой задачи идеально подходит расширение ROLLUP оператора GROUP BY. Если используемая СУБД не поддерживает это расширение, задачу можно решить, хотя и с большими трудностями, используя скалярный подзапрос или запрос с объединением (UNION).

DB2 и Oracle

Для суммирования зарплат используем функцию SUM, а результаты организовываем в подсуммы (по JOB) и общую сумму (для всей таблицы) с помощью расширения ROLLUP оператора GROUP BY:

```

1 select case grouping(job)
2         when 0 then job
3         else 'TOTAL'
4         end job,
5         sum(sal) sal
6   from emp
7  group by rollup(job)

```

SQL Server и MySQL

Для суммирования зарплат используем функцию SUM, а результаты организовываем в подсуммы (по JOB) и общую сумму (для всей таблицы) с помощью конструкции WITH ROLLUP. Затем с помощью функции COALESCE создаем заголовок 'TOTAL' для строки общей суммы (в противном случае столбец JOB этой строки будет содержать значение NULL):

```

1 select coalesce(job,'TOTAL') job,
2     sum(sal) sal
3     from emp
4     group by job with rollup

```

Для SQL Server вместо функции COALESCE уровень агрегации также можно определять с помощью функции GROUPING, показанной в рецепте для Oracle и DB2.

PostgreSQL

Аналогично решениям для SQL Server и MySQL, используем расширение ROLLUP оператора GROUP BY, но с несколько иным синтаксисом:

```

select coalesce(job,'TOTAL') job,
       sum(sal) sal
       from emp
       group by rollup(job)

```

Обсуждение

DB2 и Oracle

На первом шаге суммируем зарплаты по JOB, используя для этого агрегатную функцию SUM с группированием по JOB:

```

select job, sum(sal) sal
       from emp
       group by job

```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600

Затем с помощью расширения ROLLUP оператора GROUP BY создаем общую сумму всех зарплат вместе с подсуммами для каждой должности JOB:

```

select job, sum(sal) sal
       from emp
       group by rollup(job)

```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
	29025

В завершение применяем функцию `GROUPING` к столбцу `JOB`, чтобы создать заголовок 'TOTAL' для общей суммы. Если значение `JOB` равно `NULL`, функция `GROUPING` возвратит 1, показывая, что значение `SAL` является общей суммой, созданной расширением `ROLLUP`. Если же значение `JOB` не равно `NULL`, функция `GROUPING` возвратит 0, показывая, что значение `SAL` является результатом оператора `GROUP BY`, а не расширения `ROLLUP`. Функцию `GROUPING(JOB)` вставляем в выражение `CASE`, которое в зависимости от обстоятельств возвращает или название должности, или заголовок 'TOTAL':

```
select case grouping(job)
        when 0 then job
        else 'TOTAL'
      end job,
       sum(sal) sal
  from emp
 group by rollup(job)
```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
TOTAL	29025

SQL Server и MySQL

На первом шаге суммируем зарплаты по `JOB`, используя для этого агрегатную функцию `SUM` с группированием по `JOB`:

```
select job, sum(sal) sal
  from emp
 group by job
```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600

Затем с помощью расширения `ROLLUP` оператора `GROUP BY` создаем общую сумму всех зарплат вместе с подсуммами для каждой должности `JOB`:

```
select job, sum(sal) sal
  from emp
 group by job with rollup
```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
	29025

В завершение применяем функцию `COALESCE` к столбцу `JOB`. Если значение `JOB` равно `NULL`, значение `SAL` является общей суммой, созданной расширением `ROLLUP`. Если же значение `JOB` не равно `NULL`, значение `SAL` является результатом оператора `GROUP BY`, а не расширения `ROLLUP`:

```
select coalesce(job, 'TOTAL') job,
       sum(sal) sal
from emp
group by job with rollup
```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
TOTAL	29025

PostgreSQL

Решение для этой СУБД аналогично предшествующему решению для MySQL и SQL Server. Единственная разница — синтаксис конструкции `ROLLUP`: после `GROUP BY` указываем `ROLLUP (JOB)`.

12.13. Вычисление подсумм для всех возможных сочетаний

ЗАДАЧА

Требуется вернуть подсуммы зарплат по отделам `DEPTNO`, должностям `JOB`, а также для каждого сочетания `JOB/DEPTNO`. Также надо вернуть общую сумму всех зарплат в таблице `EMP`. Таким образом, результирующее множество должно иметь следующий вид:

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000

20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
30	MANAGER	TOTAL BY DEPT AND JOB	2850
20	MANAGER	TOTAL BY DEPT AND JOB	2975
20	ANALYST	TOTAL BY DEPT AND JOB	6000
	CLERK	TOTAL BY JOB	4150
	ANALYST	TOTAL BY JOB	6000
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600
10		TOTAL BY DEPT	8750
30		TOTAL BY DEPT	9400
20		TOTAL BY DEPT	10875
		GRAND TOTAL FOR TABLE	29025

РЕШЕНИЕ

Благодаря расширениям, добавленным в последние годы к оператору `GROUP BY`, эта задача достаточно легко поддается решению. Если используемая СУБД не поддерживает такие расширения для вычисления подсумм разных уровней, тогда их нужно вычислять вручную посредством самообъединений или скалярных подзапросов.

DB2

Для DB2 нужно выполнять приведение (`CAST`) возвращаемых функцией `GROUPING` данных к типу `CHAR(1)`:

```

1 select deptno,
2     job,
3     case cast(grouping(deptno) as char(1))||
4         cast(grouping(job) as char(1))
5         when '00' then 'TOTAL BY DEPT AND JOB'
6         when '10' then 'TOTAL BY JOB'
7         when '01' then 'TOTAL BY DEPT'
8         when '11' then 'TOTAL FOR TABLE'
9     end category,
10    sum(sal)
11 from emp
12 group by cube(deptno, job)
13 order by grouping(job), grouping(deptno)

```

Oracle

Используем расширение `CUBE` оператора `GROUP BY` вместе с оператором конкатенации `||`:

```

1 select deptno,
2     job,

```

```

3      case grouping(deptno)||grouping(job)
4          when '00' then 'TOTAL BY DEPT AND JOB'
5          when '10' then 'TOTAL BY JOB'
6          when '01' then 'TOTAL BY DEPT'
7          when '11' then 'GRAND TOTALFOR TABLE'
8      end category,
9      sum(sal) sal
10     from emp
11     group by cube(deptno,job)
12     order by grouping(job),grouping(deptno)

```

SQL Server

Используем расширение CUBE оператора GROUP BY. Возвращаемые функцией GROUPING данные нужно привести (CAST) к типу CHAR(1), а также использовать оператор конкатенации + (а не ||, как в Oracle):

```

1 select deptno,
2      job,
3      case cast(grouping(deptno)as char(1))+
4          cast(grouping(job)as char(1))
5          when '00' then 'TOTAL BY DEPT AND JOB'
6          when '10' then 'TOTAL BY JOB'
7          when '01' then 'TOTAL BY DEPT'
8          when '11' then 'GRAND TOTAL FOR TABLE'
9      end category,
10     sum(sal) sal
11     from emp
12     group by deptno,job with cube
13     order by grouping(job),grouping(deptno)

```

PostgreSQL

Решение для этой СУБД похоже на предыдущее решение, только слегка отличается от него синтаксисом оператора CUBE и конкатенацией:

```

select deptno,job
,case concat(
cast (grouping(deptno) as char(1)),cast (grouping(job) as char(1))
)
when '00' then 'TOTAL BY DEPT AND JOB'
    when '10' then 'TOTAL BY JOB'
    when '01' then 'TOTAL BY DEPT'
    when '11' then 'GRAND TOTAL FOR TABLE'
end category
, sum(sal) as sal
from emp
group by cube(deptno,job)

```

MySQL

Эта СУБД поддерживает только часть функциональности для вычисления подсумм, т. к. в ней отсутствует функция CUBE. Поэтому нужно использовать несколько объединений UNION ALL, вычисляя подсумму для каждого из них:

```

1 select deptno, job,
2     'TOTAL BY DEPT AND JOB' as category,
3     sum(sal) as sal
4   from emp
5  group by deptno, job
6  union all
7 select null, job, 'TOTAL BY JOB', sum(sal)
8   from emp
9  group by job
10 union all
11 select deptno, null, 'TOTAL BY DEPT', sum(sal)
12   from emp
13  group by deptno
14  union all
15 select null,null,'GRAND TOTAL FOR TABLE', sum(sal)
16   from emp

```

Обсуждение

Oracle, DB2 и SQL Server

Решения для этих трех СУБД практически одинаковые. На первом шаге вычисляем общие зарплаты для всех комбинаций JOB и DEPTNO, используя для этого функцию SUB и выполняя группирование как по DEPTNO, так и по JOB:

```

select deptno, job, sum(sal) sal
   from emp
  group by deptno, job

```

DEPTNO	JOB	SAL
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	CLERK	1900
20	ANALYST	6000
20	MANAGER	2975
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

Затем вычисляем подсуммы по JOB и DEPTNO вместе с общей суммой для всей таблицы. Агрегирование по SAL для DEPTNO, JOB и всей таблицы выполняем с помощью расширения CUBE оператора GROUP BY:

```
select deptno,
       job,
       sum(sal) sal
from emp
group by cube(deptno,job)
```

DEPTNO	JOB	SAL
		29025
	CLERK	4150
	ANALYST	6000
	MANAGER	8275
	SALESMAN	5600
	PRESIDENT	5000
10		8750
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20		10875
20	CLERK	1900
20	ANALYST	6000
20	MANAGER	2975
30	9400	
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

Далее оформляем результаты в более удобочитаемом виде с помощью функции `GROUPING` в сочетании с выражением `CASE`. В зависимости от происхождения значений `SAL` возвращаемое функцией `GROUPING(JOB)` значение будет 1 или 0. Если источник результата `CUBE`, то это будет значение 1, в противном случае оно будет 0. То же относится и к функции `GROUPING(DEPTNO)`. В первом шаге решения можно видеть, что группирование выполняется по `DEPTNO` и `JOB`. Следовательно, когда строка представляет сочетание и `DEPTNO`, и `JOB`, ожидаемым результатом вызовом функции `GROUPING` будет 0. Следующий запрос подтверждает это:

```
select deptno,
       job,
       grouping(deptno) is_deptno_subtotal,
       grouping(job) is_job_subtotal,
       sum(sal) sal
from emp
group by cube(deptno,job)
order by 3,4
```

DEPTNO	JOB	IS_DEPTNO_SUBTOTAL	IS_JOB_SUBTOTAL	SAL
10	CLERK	0	0	1300
10	MANAGER	0	0	2450
10	PRESIDENT	0	0	5000

20	CLERK	0	0	1900
30	CLERK	0	0	950
30	SALESMAN	0	0	5600
30	MANAGER	0	0	2850
20	MANAGER	0	0	2975
20	ANALYST	0	0	6000
10		0	1	8750
20		0	1	10875
30		0	1	9400
	CLERK	1	0	4150
	ANALYST	1	0	6000
	MANAGER	1	0	8275
	PRESIDENT	1	0	5000
	SALESMAN	1	0	5600
		1	1	29025

В завершение на основании конкатенированных значений, возвращенных функциями `GROUPING (JOB)` и `GROUPING (DEPTNO)`, с помощью выражения `CASE` определяем категорию каждой строки:

```
select deptno,
       job,
       case grouping(deptno) || grouping(job)
         when '00' then 'TOTAL BY DEPT AND JOB'
         when '10' then 'TOTAL BY JOB'
         when '01' then 'TOTAL BY DEPT'
         when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
from emp
group by cube(deptno, job)
order by grouping(job), grouping(deptno)
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
30	MANAGER	TOTAL BY DEPT AND JOB	2850
20	MANAGER	TOTAL BY DEPT AND JOB	2975
20	ANALYST	TOTAL BY DEPT AND JOB	6000
	CLERK	TOTAL BY JOB	4150
	ANALYST	TOTAL BY JOB	6000
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600

10	TOTAL BY DEPT	8750
30	TOTAL BY DEPT	9400
20	TOTAL BY DEPT	10875
	GRAND TOTAL FOR TABLE	29025

В этом решении для Oracle результаты функций `GROUPING` неявно приводятся к символьному типу, подготавливая их к конкатенации. Для СУБД DB2 и SQL Server нужно выполнять явное приведение результатов функции `GROUPING` к типу `CHAR(1)`, как показано в решениях для них. Кроме этого, в SQL Server для конкатенирования двух результатов функции `GROUPING` в одну строку надо использовать оператор конкатенирования `+`, а не `||`.

СУБД Oracle и DB2 поддерживают дополнительное, чрезвычайно полезное расширение `GROUPING SETS` оператора `GROUP BY`. Это расширение можно использовать, например, чтобы эмулировать вывод расширения `CUBE`, как показано далее. При этом для DB2 и SQL Server нужно выполнять явное приведение типов (`CAST`) таким же образом, как в решении с использованием расширения `CUBE`, чтобы обеспечить правильный формат результатов функции `GROUPING`:

```
select deptno,
       job,
       case grouping(deptno)||grouping(job)
         when '00' then 'TOTAL BY DEPT AND JOB'
         when '10' then 'TOTAL BY JOB'
         when '01' then 'TOTAL BY DEPT'
         when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
from emp
group by grouping sets ((deptno), (job), (deptno,job), ())
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950
20	ANALYST	TOTAL BY DEPT AND JOB	6000
10	MANAGER	TOTAL BY DEPT AND JOB	2450
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
	CLERK	TOTAL BY JOB	4150
	ANALYST	TOTAL BY JOB	6000
	MANAGER	TOTAL BY JOB	8275
	SALESMAN	TOTAL BY JOB	5600
	PRESIDENT	TOTAL BY JOB	5000
10		TOTAL BY DEPT	8750

20	TOTAL BY DEPT	10875
30	TOTAL BY DEPT	9400
GRAND	TOTAL FOR TABLE	29025

Расширение `GROUPING SETS` замечательно тем, что позволяет определять группы. В предыдущем запросе оно создает группы по `DEPTNO`, `JOB` и комбинации `DEPTNO` и `JOB` и, наконец, общую сумму, обозначаемую пустыми круглыми скобками. Расширение `GROUPING SETS` предоставляет серьезную гибкость в создании отчетов с разными уровнями агрегации. Например, чтобы из результатов предыдущего примера исключить общую сумму для всей таблицы, из `GROUPING SETS` нужно просто убрать пустые круглые скобки:

```
/* без общей суммы */
```

```
select deptno,
       job,
       case grouping(deptno)||grouping(job)
         when '00' then 'TOTAL BY DEPT AND JOB'
         when '10' then 'TOTAL BY JOB'
         when '01' then 'TOTAL BY DEPT'
         when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
from emp
group by grouping sets ((deptno), (job), (deptno, job))
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950
20	ANALYST	TOTAL BY DEPT AND JOB	6000
10	MANAGER	TOTAL BY DEPT AND JOB	2450
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
	CLERK	TOTAL BY JOB	4150
	ANALYST	TOTAL BY JOB	6000
	MANAGER	TOTAL BY JOB	8275
	SALESMAN	TOTAL BY JOB	5600
	PRESIDENT	TOTAL BY JOB	5000
10		TOTAL BY DEPT	8750
20		TOTAL BY DEPT	10875
30		TOTAL BY DEPT	9400

Также из результирующего множества можно исключить какую-либо подсумму — например, по `DEPTNO`, просто удалив `(DEPTNO)` из оператора `GROUPING SETS`:

/* без подсумм по DEPTNO */

```
select deptno,
       job,
       case grouping(deptno)||grouping(job)
         when '00' then 'TOTAL BY DEPT AND JOB'
         when '10' then 'TOTAL BY JOB'
         when '01' then 'TOTAL BY DEPT'
         when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
from emp
group by grouping sets ((job),(deptno,job),())
order by 3
```

DEPTNO	JOB	CATEGORY	SAL
		GRAND TOTAL FOR TABLE	29025
10	CLERK	TOTAL BY DEPT AND JOB	1300
20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950
20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
10	MANAGER	TOTAL BY DEPT AND JOB	450
	CLERK	TOTAL BY JOB	4150
	SALESMAN	TOTAL BY JOB	5600
	PRESIDENT	TOTAL BY JOB	5000
	MANAGER	TOTAL BY JOB	8275
	ANALYST	TOTAL BY JOB	6000

Как можно видеть, благодаря расширению GROUPING SETS можно легко манипулировать подсуммами и общей суммой, представляя данные в разных вариантах.

MySQL

На первом шаге суммируем зарплаты по DEPTNO и JOB, используя для этого агрегатную функцию SUM с группированием по этим обоим столбцам:

```
select deptno, job,
       'TOTAL BY DEPT AND JOB' as category,
       sum(sal) as sal
from emp
group by deptno, job
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000


```

20 CLERK      TOTAL BY DEPT AND JOB    1900
20 ANALYST   TOTAL BY DEPT AND JOB    6000
20 MANAGER   TOTAL BY DEPT AND JOB    2975
30 CLERK     TOTAL BY DEPT AND JOB     950
30 MANAGER   TOTAL BY DEPT AND JOB    2850
30 SALESMAN  TOTAL BY DEPT AND JOB    5600

```

Затем вычисляем подсуммы зарплат по должностям (JOB), используя для этого UNION ALL:

```

select deptno, job,
       'TOTAL BY DEPT AND JOB' as category,
       sum(sal) as sal
  from emp
 group by deptno, job
 union all
select null, job, 'TOTAL BY JOB', sum(sal)
  from emp
 group by job

```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	CLERK	TOTAL BY DEPT AND JOB	950
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
	ANALYST	TOTAL BY JOB	6000
	CLERK	TOTAL BY JOB	4150
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600

Далее вычисляем подсуммы зарплат по отделам (DEPTNO), опять используя для этого UNION ALL:

```

select deptno, job,
       'TOTAL BY DEPT AND JOB' as category,
       sum(sal) as sal
  from emp
 group by deptno, job
 union all
select null, job, 'TOTAL BY JOB', sum(sal)
  from emp
 group by job
 union all

```

```
select deptno, null, 'TOTAL BY DEPT', sum(sal)
  from emp
  group by deptno
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	CLERK	TOTAL BY DEPT AND JOB	950
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
	ANALYST	TOTAL BY JOB	6000
	CLERK	TOTAL BY JOB	4150
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600
10		TOTAL BY DEPT	8750
20		TOTAL BY DEPT	10875
30		TOTAL BY DEPT	9400

В завершение вычисляем общую сумму зарплат для всей таблицы, снова используя для этого UNION ALL:

```
select deptno, job,
  'TOTAL BY DEPT AND JOB' as category,
  sum(sal) as sal
  from emp
  group by deptno, job
  union all
select null, job, 'TOTAL BY JOB', sum(sal)
  from emp
  group by job
  union all
select deptno, null, 'TOTAL BY DEPT', sum(sal)
  from emp
  group by deptno
  union all
select null,null, 'GRAND TOTAL FOR TABLE', sum(sal)
  from emp
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000

20	CLERK	TOTAL BY DEPT AND JOB	1900
20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	CLERK	TOTAL BY DEPT AND JOB	950
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
	ANALYST	TOTAL BY JOB	6000
	CLERK	TOTAL BY JOB	4150
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600
10		TOTAL BY DEPT	8750
20		TOTAL BY DEPT	10875
30		TOTAL BY DEPT	9400
		GRAND TOTAL FOR TABLE	29025

12.14. Выделение строк, не содержащих подсумм

ЗАДАЧА

Предположим, что мы создали отчет с помощью расширения CUBE оператора GROUP BY, и нам нужно каким-то способом отличить строки, созданные обычным оператором GROUP BY, от строк, созданных расширениями CUBE или ROLLUP.

Далее приводится результирующее множество запроса, в котором зарплаты таблицы EMP разбиты по разным категориям подсумм с помощью расширения CUBE оператора GROUP BY:

DEPTNO	JOB	SAL
		29025
	CLERK	4150
	ANALYST	6000
	MANAGER	8275
	SALESMAN	5600
	PRESIDENT	5000
10		8750
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20		10875
20	CLERK	1900
20	ANALYST	6000
20	MANAGER	2975
30		9400
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

Этот вывод содержит подсуммы всех зарплат по DEPTNO и JOB (по каждому значению JOB для DEPTNO), подсуммы всех зарплат по DEPTNO, подсуммы всех зарплат по JOB и, наконец, общую сумму всех зарплат в таблице EMP. Требуется четко обозначить разные уровни агрегации — определить категорию, к которой относится каждое агрегированное значение (представляет ли это значение в столбце SAL подсумму по DEPTNO, JOB или общую сумму). То есть требуется получить следующее результирующее множество:

DEPTNO	JOB	SAL	DEPTNO_SUBTOTALS	JOB_SUBTOTALS
		29025	1	1
	CLERK	4150	1	0
	ANALYST	6000	1	0
	MANAGER	8275	1	0
	SALESMAN	5600	1	0
	PRESIDENT	5000	1	0
10		8750	0	1
10	CLERK	1300	0	0
10	MANAGER	2450	0	0
10	PRESIDENT	5000	0	0
20		10875	0	1
20	CLERK	1900	0	0
20	ANALYST	6000	0	0
20	MANAGER	2975	0	0
30		9400	0	1
30	CLERK	950	0	0
30	MANAGER	2850	0	0
30	SALESMAN	5600	0	0

Цель этого рецепта — пояснить использование CUBE и GROUPING при вычислении подсумм. На момент подготовки материала этой книги MySQL не поддерживает ни CUBE, ни GROUPING.

РЕШЕНИЕ

Чтобы определить, является ли значение подсуммой, созданной расширениями CUBE или ROLLUP, или же *суперагрегатным* значением, используем функцию GROUPING. Далее приводится пример решения для PostgreSQL, DB2 и Oracle:

```

1 select deptno, jo) sal,
2     grouping(deptno) deptno_subtotals,
3     grouping(job) job_subtotals
4   from emp
5  group by cube(deptno,job)

```

Решение для SQL Server отличается от приведенного решения только оформлением оператора CUBE:

```

1 select deptno, job, sum(sal) sal,
2     grouping(deptno) deptno_subtotals,
3     grouping(job) job_subtotals

```

```

4   from emp
5   group by deptno,job with cube

```

Обсуждение

Если значение `DEPTNO_SUBTOTALS` равно 0, а `JOB_SUBTOTALS` равно 1 (в таком случае значение `JOB` равно `NULL`), значение `SAL` представляет собой подсумму зарплат по `DEPTNO`, созданную с помощью `CUBE`. Если значение `JOB_SUBTOTALS` равно 0, а `DEPTNO_SUBTOTALS` равно 1 (в таком случае значение `DEPTNO` равно `NULL`), значение `SAL` представляет собой подсумму зарплат по `JOB`, созданную с помощью `CUBE`. Строки, в которых как `DEPTNO_SUBTOTALS`, так и `JOB_SUBTOTALS` равно 0, представляют результат обычной агрегации (сумму `SAL` для каждого сочетания `DEPTNO/JOB`).

12.15. Маркировка строк с помощью выражений CASE

ЗАДАЧА

Требуется обозначить значения столбца — скажем, столбца `JOB` таблицы `EMP`, последовательностью «логических» флагов. Например, получить следующее результирующее множество:

ENAME	IS_CLERK	IS_SALES	IS_MGR	IS_ANALYST	IS_PREZ
KING	0	0	0	0	1
SCOTT	0	0	0	1	0
FORD	0	0	0	1	0
JONES	0	0	1	0	0
BLAKE	0	0	1	0	0
CLARK	0	0	1	0	0
ALLEN	0	1	0	0	0
WARD	0	1	0	0	0
MARTIN	0	1	0	0	0
TURNER	0	1	0	0	0
SMITH	1	0	0	0	0
MILLER	1	0	0	0	0
ADAMS	1	0	0	0	0
JAMES	1	0	0	0	0

Такое результирующее множество может быть полезным для более глубокого понимания данных, предоставляя возможность рассмотрения их под другим углом, чем в обычном результирующем множестве.

РЕШЕНИЕ

С помощью выражения `CASE` определяем должность каждого служащего и возвращаем 1 или 0 для ее обозначения. Выражение `CASE` нужно применять к каждой возможной должности (`JOB`), создавая, таким образом, для нее отдельный столбец:

```

1 select ename,
2     case when job = 'CLERK'
3         then 1 else 0
4     end as is_clerk,
5     case when job = 'SALESMAN'
6         then 1 else 0
7     end as is_sales,
8     case when job = 'MANAGER'
9         then 1 else 0
10    end as is_mgr,
11    case when job = 'ANALYST'
12        then 1 else 0
13    end as is_analyst,
14    case when job = 'PRESIDENT'
15        then 1 else 0
16    end as is_prez
17 from emp
18 order by 2,3,4,5,6

```

Обсуждение

Код решения по большому счету не требует объяснения. Но в случае проблем с пониманием просто добавьте столбец `JOB` в оператор `SELECT`:

```

select ename,
       job,
       case when job = 'CLERK'
           then 1 else 0
       end as is_clerk,
       case when job = 'SALESMAN'
           then 1 else 0
       end as is_sales,
       case when job = 'MANAGER'
           then 1 else 0
       end as is_mgr,
       case when job = 'ANALYST'
           then 1 else 0
       end as is_analyst,
       case when job = 'PRESIDENT'
           then 1 else 0
       end as is_prez
from emp
order by 2

```

ENAME	JOB	IS_CLERK	IS_SALES	IS_MGR	IS_ANALYST	IS_PREZ
SCOTT	ANALYST	0	0	0	1	0
FORD	ANALYST	0	0	0	1	0
SMITH	CLERK	1	0	0	0	0

ADAMS	CLERK	1	0	0	0	0
MILLER	CLERK	1	0	0	0	0
JAMES	CLERK	1	0	0	0	0
JONES	MANAGER	0	0	1	0	0
CLARK	MANAGER	0	0	1	0	0
BLAKE	MANAGER	0	0	1	0	0
KING	PRESIDENT	0	0	0	0	1
ALLEN	SALESMAN	0	1	0	0	0
MARTIN	SALESMAN	0	1	0	0	0
TURNER	SALESMAN	0	1	0	0	0
WARD	SALESMAN	0	1	0	0	0

12.16. Создание разреженной матрицы

ЗАДАЧА

Требуется создать разреженную матрицу следующего вида, в которой транспонированы столбцы DEPTNO и JOB таблицы EMP:

D10	D20	D30	CLERKS	MGRS	PREZ	ANALS	SALES
	SMITH		SMITH				
		ALLEN					ALLEN
		WARD					WARD
	JONES			JONES			
		MARTIN					MARTIN
		BLAKE		BLAKE			
CLARK				CLARK			
	SCOTT					SCOTT	
KING					KING		
		TURNER					TURNER
	ADAMS		ADAMS				
		JAMES	JAMES				
	FORD					FORD	
MILLER			MILLER				

РЕШЕНИЕ

Для создания разреженной трансформации строк в столбцы используем выражения CASE:

```

1 select case deptno when 10 then ename end as d10,
2         case deptno when 20 then ename end as d20,
3         case deptno when 30 then ename end as d30,
4         case job when 'CLERK' then ename end as clerks,
5         case job when 'MANAGER' then ename end as mgrs,
6         case job when 'PRESIDENT' then ename end as prez,
7         case job when 'ANALYST' then ename end as anals,
8         case job when 'SALESMAN' then ename end as sales
9 from emp
```

Обсуждение

Чтобы транспонировать значения DEPTNO и JOB строк в столбцы, просто проверяем возможные значения, возвращаемые этими строками, с помощью выражения CASE. Вот и всего делов-то. Кстати, чтобы «уплотнить» отчет, поудаляв из него строки со значениями NULL, нужно определиться со столбцом, по которому выполнять группирование. Например, некоторые значения NULL можно удалить, выполнив ранжирование служащих в каждом отделе с помощью оконной функции ROW_NUMBER OVER, а затем применив агрегатную функцию MAX:

```
select max(case deptno when 10 then ename end) d10,
       max(case deptno when 20 then ename end) d20,
       max(case deptno when 30 then ename end) d30,
       max(case job when 'CLERK' then ename end) clerks,
       max(case job when 'MANAGER' then ename end) mgrs,
       max(case job when 'PRESIDENT' then ename end) prez,
       max(case job when 'ANALYST' then ename end) anals,
       max(case job when 'SALESMAN' then ename end) sales
  from (
select deptno, job, ename,
       row_number()over(partition by deptno order by empno) rn
  from emp
   ) x
 group by rn
```

D10	D20	D30	CLERKS	MGRS	PREZ	ANALS	SALES
CLARK	SMITH	ALLEN	SMITH	CLARK			ALLEN
KING	JONES	WARD		JONES	KING		WARD
MILLER	SCOTT	MARTIN	MILLER			SCOTT	MARTIN
	ADAMS	BLAKE	ADAMS	BLAKE			
	FORD	TURNER				FORD	TURNER
		JAMES	JAMES				

12.17. Группирование строк по интервалам времени

ЗАДАЧА

Требуется обобщить данные по некоторому интервалу времени. Например, по данным журнала транзакций надо подсчитать количество транзакций, выполненных в каждый пятисекундный интервал. Таблица журнала транзакций TRX_LOG содержит следующие строки:

```
select trx_id,
       trx_date,
       trx_cnt
  from trx_log
```


TRX_ID	TRX_DATE	TRX_CNT
1	28-JUL-2020 19:03:07	44
2	28-JUL-2020 19:03:08	18
3	28-JUL-2020 19:03:09	23
4	28-JUL-2020 19:03:10	29
5	28-JUL-2020 19:03:11	27
6	28-JUL-2020 19:03:12	45
7	28-JUL-2020 19:03:13	45
8	28-JUL-2020 19:03:14	32
9	28-JUL-2020 19:03:15	41
10	28-JUL-2020 19:03:16	15
11	28-JUL-2020 19:03:17	24
12	28-JUL-2020 19:03:18	47
13	28-JUL-2020 19:03:19	37
14	28-JUL-2020 19:03:20	48
15	28-JUL-2020 19:03:21	46
16	28-JUL-2020 19:03:22	44
17	28-JUL-2020 19:03:23	36
18	28-JUL-2020 19:03:24	41
19	28-JUL-2020 19:03:25	33
20	28-JUL-2020 19:03:26	19

Результирующее множество должно иметь следующий вид:

GRP	TRX_START	TRX_END	TOTAL
1	28-JUL-2020 19:03:07	28-JUL-2020 19:03:11	141
2	28-JUL-2020 19:03:12	28-JUL-2020 19:03:16	178
3	28-JUL-2020 19:03:17	28-JUL-2020 19:03:21	202
4	28-JUL-2020 19:03:22	28-JUL-2020 19:03:26	173

РЕШЕНИЕ

На первом шаге разобьем записи на пять блоков. Такое логическое группирование можно осуществить несколькими способами. Здесь мы воспользуемся методом из *рецепта 12.7*, выполняя деление значений `TRX_ID` на пять.

Создав «группы», используем агрегатные функции `MIN`, `MAX` и `SUM`, чтобы вычислить время начала и конца транзакций каждой «группы» и количество ее транзакций (для `SQL Server` вместо функции `CEIL` нужно использовать функцию `CEILING`):

```

1 select ceil(trx_id/5.0) as grp,
2     min(trx_date) as trx_start,
3     max(trx_date) as trx_end,
4     sum(trx_cnt) as total
5   from trx_log
6  group by ceil(trx_id/5.0)

```

Обсуждение

Первый шаг и ключ к полному решению — разбить строки на логические группы. Для этого мы делим идентификатор строки `TRX_ID` на пять и округляем частное в большую сторону. Например:

```
select trx_id,
       trx_date,
       trx_cnt,
       trx_id/5.0 as val,
       ceil(trx_id/5.0) as grp
from   trx_log
```

TRX_ID	TRX_DATE		TRX_CNT	VAL	GRP
1	28-JUL-2020 19:03:07		44	.20	1
2	28-JUL-2020 19:03:08		18	.40	1
3	28-JUL-2020 19:03:09		23	.60	1
4	28-JUL-2020 19:03:10		29	.80	1
5	28-JUL-2020 19:03:11		27	1.00	1
6	28-JUL-2020 19:03:12		45	1.20	2
7	28-JUL-2020 19:03:13		45	1.40	2
8	28-JUL-2020 19:03:14		32	1.60	2
9	28-JUL-2020 19:03:15		41	1.80	2
10	28-JUL-2020 19:03:16		15	2.00	2
11	28-JUL-2020 19:03:17		24	2.20	3
12	28-JUL-2020 19:03:18		47	2.40	3
13	28-JUL-2020 19:03:19		37	2.60	3
14	28-JUL-2020 19:03:20		48	2.80	3
15	28-JUL-2020 19:03:21		46	3.00	3
16	28-JUL-2020 19:03:22		44	3.20	4
17	28-JUL-2020 19:03:23		36	3.40	4
18	28-JUL-2020 19:03:24		41	3.60	4
19	28-JUL-2020 19:03:25		33	3.80	4
20	28-JUL-2020 19:03:26		19	4.00	4

В завершение с помощью агрегатных функций вычисляем общее количество транзакций каждые пять секунд, а также время начала и завершения каждой транзакции:

```
select ceil(trx_id/5.0) as grp,
       min(trx_date) as trx_start,
       max(trx_date) as trx_end,
       sum(trx_cnt) as total
from   trx_log
group by ceil(trx_id/5.0)
```

GRP	TRX_START		TRX_END		TOTAL
1	28-JUL-2020 19:03:07		28-JUL-2020 19:03:11		141
2	28-JUL-2020 19:03:12		28-JUL-2020 19:03:16		178

```

3 28-JUL-2020 19:03:17 28-JUL-2005 19:03:21      202
4 28-JUL-2020 19:03:22 28-JUL-2005 19:03:26      173

```

Данные, слегка отличающиеся от рассмотренных в примере (например, строки не имеют ID), можно всегда разбить на подобные «группы», разделив на пять значение секунд каждой строки `TRX_DATE`. Затем для каждого значения `TRX_DATE` можно добавить значение часов и выполнить группирование по фактическому часу и логической группе `GRP`. Далее представлен пример реализации этого метода с использованием функций `TO_CHAR` и `TO_NUMBER` СУБД Oracle. Для других СУБД нужно использовать соответствующие функции для работы с датами и форматирования символов:

```

select trx_date,trx_cnt,
       to_number(to_char(trx_date,'hh24')) hr,
       ceil(to_number(to_char(trx_date-1/24/60/60,'miss'))/5.0) grp
from   trx_log

```

TRX_DATE	TRX_CNT	HR	GRP
28-JUL-2020 19:03:07	44	19	62
28-JUL-2020 19:03:08	18	19	62
28-JUL-2020 19:03:09	23	19	62
28-JUL-2020 19:03:10	29	19	62
28-JUL-2020 19:03:11	27	19	62
28-JUL-2020 19:03:12	45	19	63
28-JUL-2020 19:03:13	45	19	63
28-JUL-2020 19:03:14	32	19	63
28-JUL-2020 19:03:15	41	19	63
28-JUL-2020 19:03:16	15	19	63
28-JUL-2020 19:03:17	24	19	64
28-JUL-2020 19:03:18	47	19	64
28-JUL-2020 19:03:19	37	19	64
28-JUL-2020 19:03:20	48	19	64
28-JUL-2020 19:03:21	46	19	64
28-JUL-2020 19:03:22	44	19	65
28-JUL-2020 19:03:23	36	19	65
28-JUL-2020 19:03:24	41	19	65
28-JUL-2020 19:03:25	33	19	65
28-JUL-2020 19:03:26	19	19	65

Ключевой момент здесь заключается в создании одной группы для каждого пятисекундного интервала, независимо от фактических значений `GRP`. После этого можно применять агрегатные функции таким же образом, как и в первоначальном решении:

```

select hr,grp,sum(trx_cnt) total
from (
select trx_date,trx_cnt,
       to_number(to_char(trx_date,'hh24')) hr,
       ceil(to_number(to_char(trx_date-1/24/60/60,'miss'))/5.0) grp

```

```

from trx_log
) x
group by hr,grp

```

HR	GRP	TOTAL
19	62	141
19	63	178
19	64	202
19	65	173

Включение часа в группы полезно в том случае, когда журнал транзакций охватывает несколько часов. В DB2 и Oracle такой же результат можно получить, используя оконную функцию `SUM OVER`. Следующий запрос возвращает все строки таблицы `TRX_LOG` и промежуточную сумму `RUNNING TOTAL` транзакций (столбец `TRX_CTN`) по логическим «группам», а также общую сумму `TOTAL` транзакций (столбец `TRX_CNT`) в каждой строке в «группе»:

```

select trx_id, trx_date, trx_cnt,
       sum(trx_cnt)over(partition by ceil(trx_id/5.0)
                        order by trx_date
                        range between unbounded preceding
                        and current row) runing_total,
       sum(trx_cnt)over(partition by ceil(trx_id/5.0)) total,
       case when mod(trx_id,5.0) = 0 then 'X' end grp_end
from trx_log

```

TRX_ID	TRX_DATE	TRX_CNT	RUNING_TOTAL	TOTAL	GRP_END
1	28-JUL-2020 19:03:07	44	44	141	
2	28-JUL-2020 19:03:08	18	62	141	
3	28-JUL-2020 19:03:09	23	85	141	
4	28-JUL-2020 19:03:10	29	114	141	
5	28-JUL-2020 19:03:11	27	141	141	X
6	28-JUL-2020 19:03:12	45	45	178	
7	28-JUL-2020 19:03:13	45	90	178	
8	28-JUL-2020 19:03:14	32	122	178	
9	28-JUL-2020 19:03:15	41	163	178	
10	28-JUL-2020 19:03:16	15	178	178	X
11	28-JUL-2020 19:03:17	24	24	202	
12	28-JUL-2020 19:03:18	47	71	202	
13	28-JUL-2020 19:03:19	37	108	202	
14	28-JUL-2020 19:03:20	48	156	202	
15	28-JUL-2020 19:03:21	46	202	202	X
16	28-JUL-2020 19:03:22	44	44	173	
17	28-JUL-2020 19:03:23	36	80	173	
18	28-JUL-2020 19:03:24	41	121	173	
19	28-JUL-2020 19:03:25	33	154	173	
20	28-JUL-2020 19:03:26	19	173	173	X

12.18. Одновременная агрегация разных групп/сегментов

ЗАДАЧА

Требуется выполнить агрегацию одновременно в нескольких измерениях. Например, надо вернуть результирующее множество, содержащее имена всех служащих, их отделы, количество служащих в отделе каждого служащего (включая его самого), количество служащих с такой же должностью (включая этого служащего), а также общее количество служащих в таблице EMP. Результирующее множество должно выглядеть следующим образом:

ENAME	DEPTNO	DEPTNO_CNT	JOB	JOB_CNT	TOTAL
MILLER	10	3	CLERK	4	14
CLARK	10	3	MANAGER	3	14
KING	10	3	PRESIDENT	1	14
SCOTT	20	5	ANALYST	2	14
FORD	20	5	ANALYST	2	14
SMITH	20	5	CLERK	4	14
JONES	20	5	MANAGER	3	14
ADAMS	20	5	CLERK	4	14
JAMES	30	6	CLERK	4	14
MARTIN	30	6	SALESMAN	4	14
TURNER	30	6	SALESMAN	4	14
WARD	30	6	SALESMAN	4	14
ALLEN	30	6	SALESMAN	4	14
BLAKE	30	6	MANAGER	3	14

РЕШЕНИЕ

Используем функцию `COUNT OVER`, указывая в параметре разные сегменты или группы данных, по которым нужно выполнить агрегацию:

```
select ename,
       deptno,
       count(*)over(partition by deptno) deptno_cnt,
       job,
       count(*)over(partition by job) job_cnt,
       count(*)over() total
from emp
```

Обсуждение

Этот пример убедительно демонстрирует мощь и удобство использования оконных функций. Просто указав разные сегменты или группы данных, над которыми нужно выполнить агрегацию, можно создавать чрезвычайно подробные отчеты без необходимости выполнять множественные самообъединения и размещения в списке `SELECT` громоздких и, возможно, низкопроизводительных подзапросов. Вся

работа выполняется функцией `COUNT OVER`. Чтобы понять создаваемый этой функцией вывод, рассмотрим более подробно оператор `OVER` каждой операции `COUNT`:

```
count(*)over(partition by deptno)
```

```
count(*)over(partition by job)
```

```
count(*)over()
```

Вспомним основные части оператора `OVER`: конструкцию `PARTITION BY`, определяющую сегменты, и конструкцию `ORDER BY`, определяющую логическое упорядочивание. Посмотрим на первый оператор `COUNT`, который выполняет сегментирование по значениям `DEPTNO`. Строки таблицы `EMP` будут сгруппированы по значениям `DEPTNO`, и оператор `COUNT` подсчитает все строки в каждой группе. Поскольку кадр или окно не определены (отсутствует оператор `ORDER BY`), подсчитываются все строки в группе. Оператор `PARTITION BY` находит все уникальные значения `DEPTNO`, после чего функция `COUNT` подсчитывает количество строк с каждым из этих значений. В конкретном примере `COUNT(*)OVER(PARTITION BY DEPTNO)` оператор `PARTITION BY` определяет сегменты или группы значений 10, 20 и 30.

Такую же обработку выполняет и вторая функция `COUNT`, в которой сегментирование осуществляется по значениям `JOB`. Последняя функция `COUNT` не осуществляет никакого сегментирования, а просто содержит пустые круглые скобки, что означает «вся таблица». Таким образом, тогда как первые две операции `COUNT` агрегируют значения для определенных групп или сегментов, последняя из них подсчитывает все строки в таблице `EMP`.



Важно иметь в виду, что оконные функции выполняются после обработки выражения `WHERE`. Если результирующее множество отфильтровать каким-либо образом — например, исключив всех служащих отдела 10, тогда общее значение `TOTAL` будет не 14, а 11. Чтобы отфильтровать результаты после применения оконной функции, ее необходимо вставить во вложенный запрос и отфильтровать уже результаты, возвращенные этим запросом.

12.19. Агрегирования скользящего диапазона значений

ЗАДАЧА

Требуется вычислить скользящую агрегацию — например, скользящую сумму зарплат в таблице `EMP`. В частности, надо вычислить сумму для каждого периода длительностью 90 дней, начиная с даты приема на работу `HIREDATE` первого служащего. Мы хотим увидеть колебание расходов для каждого 90-дневного периода между первым и последним принятыми на работу служащими. Результирующее множество должно иметь следующий вид:

HIREDATE	SAL	SPENDING_PATTERN
17-DEC-2010	800	800
20-FEB-2011	1600	2400
22-FEB-2011	1250	3650

02-APR-2011	2975	5825
01-MAY-2011	2850	8675
09-JUN-2011	2450	8275
08-SEP-2011	1500	1500
28-SEP-2011	1250	2750
17-NOV-2011	5000	7750
03-DEC-2011	950	11700
03-DEC-2011	3000	11700
23-JAN-2012	1300	10250
09-DEC-2012	3000	3000
12-JAN-2013	1100	4100

РЕШЕНИЕ

Если используемая СУБД поддерживает возможность задания скользящего окна в операторе кадрирования оконных функций, эта задача очень легко поддается решению. Ключ к решению — выполнить упорядочивание по `HIREDATE` в оконной функции, а затем задать окно размером в 90 дней, начиная с первого принятого на работу служащего. Сумма будет вычисляться по зарплатам служащих, принятых на работу в течение 90 дней до даты `HIREDATE` текущего служащего (зарплата которого также включается в сумму). Если используемая СУБД не поддерживает оконные функции, можно использовать скалярные подзапросы, но решение будет более сложным.

DB2 и Oracle

Для этих СУБД используем оконную функцию `SUM OVER` и выполняем упорядочивание по `HIREDATE`. Для вычисления суммы зарплат всех служащих, принятых на работу в течение предыдущих 90 дней, включая зарплату самого служащего, в операторе сегментирования задаем период длительностью в 90 дней. Поскольку DB2 не позволяет задать `HIREDATE` в операторе `ORDER BY` оконной функции (строка 3 в следующем коде), вместо этого упорядочивание можно выполнить по `DAYS(HIREDATE)`:

```

1 select hiredate,
2       sal,
3       sum(sal)over(order by days(hiredate)
4                   range between 90 preceding
5                   and current row) spending_pattern
6 from emp e
```

Решение для Oracle более прямолинейно, благодаря поддержке сортировки по данным типа `datetime` его оконными функциями:

```

1 select hiredate,
2       sal,
3       sum(sal)over(order by hiredate
4                   range between 90 preceding
5                   and current row) spending_pattern
6 from emp e
```

MySQL

Используем такую же оконную функцию, но с немного другим синтаксисом:

```

1 select hiredate,
2     sal,
3     sum(sal)over(order by hiredate
4                 range interval 90 day preceding ) spending_pattern
5 from emp e

```

PostgreSQL и SQL Server

Вычисляем сумму зарплат всех служащих, принятых на работу в течение 90 дней до приема на работу текущего служащего, включая зарплату самого служащего, с помощью скалярного подзапроса:

```

1 select e.hiredate,
2     e.sal,
3     (select sum(sal) from emp d
4     where d.hiredate between e.hiredate-90
5     and e.hiredate) as spending_pattern
6 from emp e
7 order by 1

```

Обсуждение

DB2, MySQL и Oracle

Решения для DB2, MySQL и Oracle логически одинаковые. Единственная небольшая разница между этими решениями состоит в способе задания `HIREDATE` в операторе `ORDER BY` оконной функции и в синтаксисе кода SQL для задания временного интервала. На момент подготовки материала этой книги DB2 не позволяет использовать значения типа `DATE` в таком операторе `ORDER BY` при задании размера окна посредством числового значения. Например, конструкция

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

позволяет выполнять упорядочивание по дате, а вот конструкция

```
RANGE BETWEEN 90 PRECEDING AND CURRENT ROW
```

не позволяет.

Чтобы понять, как работает запрос решения, нужно просто понимать, что делает оконная функция. В задаваемом нами окне данных зарплаты всех служащих упорядочиваются по `HIREDATE`. Затем функция суммирует зарплаты. Но при этом суммируются не все зарплаты, а происходит следующее:

1. Обрабатывается зарплата первого принятого на работу служащего. Так как до него не было никаких других приемов на работу, на этом этапе сумма состоит только из зарплаты первого служащего.

2. Обрабатывается зарплата следующего (по `HIREDATE`) принятого на работу служащего. Зарплата этого служащего включается в скользящую сумму вместе с зарплатами всех других служащих, принятых на работу в течение 90 дней до найма рассматриваемого служащего.

Значение `HIREDATE` для первого служащего — 17 декабря 2010 г., а следующего принятого на работу служащего — 20 февраля 2011 г. Второй служащий был принят на работу меньше чем через 90 дней после первого служащего, и поэтому скользящая сумма для него будет 2400 (1600 + 800). Если вам непонятно, откуда берутся значения в `SPENDING_PATTERN`, внимательно изучите следующий запрос и его результирующее множество:

```
select distinct
    dense_rank() over (order by e.hiredate) window,
    e.hiredate current_hiredate,
    d.hiredate hiredate_within_90_days,
    d.sal sals_used_for_sum
from emp e,
     emp d
where d.hiredate between e.hiredate-90 and e.hiredate
```

```
WINDOW CURRENT_HIREDATE HIREDATE_WITHIN_90_DAYS SALS_USED_FOR_SUM
```

	WINDOW	CURRENT_HIREDATE	HIREDATE_WITHIN_90_DAYS	SALS_USED_FOR_SUM
1	17-DEC-2010	17-DEC-2010		800
2	20-FEB-2011	17-DEC-2010		800
2	20-FEB-2011	20-FEB-2011		1600
3	22-FEB-2011	17-DEC-2010		800
3	22-FEB-2011	20-FEB-2011		1600
3	22-FEB-2011	22-FEB-2011		1250
4	02-APR-2011	20-FEB-2011		1600
4	02-APR-2011	22-FEB-2011		1250
4	02-APR-2011	02-APR-2011		2975
5	01-MAY-2011	20-FEB-2011		1600
5	01-MAY-2011	22-FEB-2011		1250
5	01-MAY-2011	02-APR-2011		2975
5	01-MAY-2011	01-MAY-2011		2850
6	09-JUN-2011	02-APR-2011		2975
6	09-JUN-2011	01-MAY-2011		2850
6	09-JUN-2011	09-JUN-2011		2450
7	08-SEP-2011	08-SEP-2011		1500
8	28-SEP-2011	08-SEP-2011		1500
8	28-SEP-2011	28-SEP-2011		1250
9	17-NOV-2011	08-SEP-2011		1500
9	17-NOV-2011	28-SEP-2011		1250
9	17-NOV-2011	17-NOV-2011		5000
10	03-DEC-2011	08-SEP-2011		1500
10	03-DEC-2011	28-SEP-2011		1250
10	03-DEC-2011	17-NOV-2011		5000
10	03-DEC-2011	03-DEC-2011		950

10	03-DEC-2011	03-DEC-2011	3000
11	23-JAN-2012	17-NOV-2011	5000
11	23-JAN-2012	03-DEC-2011	950
11	23-JAN-2012	03-DEC-2011	3000
11	23-JAN-2012	23-JAN-2012	1300
12	09-DEC-2012	09-DEC-2012	3000
13	12-JAN-2013	09-DEC-2012	3000
13	12-JAN-2013	12-JAN-2013	1100

Как можно видеть в столбце WINDOW, только строки с одинаковыми значениями WINDOW учитываются в каждой сумме. Возьмем, например, значение WINDOW 3. Для вычисления суммы этого окна использовались зарплаты 800, 1600 и 1250, сумма которых составляет 3650. В конечном результирующем множестве в разд. «Задача» можно видеть, что значение SPENDING_PATTERN для 22 февраля 2011 (WINDOW 3) равно 3650. Чтобы убедиться в том, что предыдущее самообъединение содержит правильные зарплаты для заданных окон, просто суммируем значения в столбце SALS_USED_FOR_SUM и выполняем группировку по CURRENT_DATE. Результат должен быть таким же, как и результирующее множество в разд. «Задача» (без дубликата строки для 3 декабря 2011 г.):

```
select current_hiredate,
       sum(sals_used_for_sum) spending_pattern
  from (
select distinct
       dense_rank()over(order by e.hiredate) window,
       e.hiredate current_hiredate,
       d.hiredate hiredate_within_90_days,
       d.sal sals_used_for_sum
  from emp e,
       emp d
 where d.hiredate between e.hiredate-90 and e.hiredate
       ) x
 group by current_hiredate
```

```
CURRENT_HIREDATE SPENDING_PATTERN
```

```
-----
17-DEC-2010          800
20-FEB-2011        2400
22-FEB-2011        3650
02-APR-2011        5825
01-MAY-2011        8675
09-JUN-2011        8275
08-SEP-2011        1500
28-SEP-2011        2750
17-NOV-2011        7750
03-DEC-2011       11700
23-JAN-2012       10250
09-DEC-2012        3000
12-JAN-2013        4100
```

PostgreSQL и SQL Server

Ключ к решению — использование скалярного подзапроса (можно использовать и самообъединение) с агрегатной функцией `sum`. Получить лучшее представление о работе решения можно, просто преобразовав его в самообъединение и посмотрев, какие строки участвуют в вычислениях. Рассмотрим следующий запрос, который возвращает такое же результирующее множество, что и предыдущее решение:

```
select e.hiredate,
       e.sal,
       sum(d.sal) as spending_pattern
  from emp e, emp d
 where d.hiredate
        between e.hiredate-90 and e.hiredate
 group by e.hiredate,e.sal
 order by 1
```

HIREDATE	SAL	SPENDING_PATTERN
17-DEC-2010	800	800
20-FEB-2011	1600	2400
22-FEB-2011	1250	3650
02-APR-2011	2975	5825
01-MAY-2011	2850	8675
09-JUN-2011	2450	8275
08-SEP-2011	1500	1500
28-SEP-2011	1250	2750
17-NOV-2011	5000	7750
03-DEC-2011	950	11700
03-DEC-2011	3000	11700
23-JAN-2012	1300	10250
09-DEC-2012	3000	3000
12-JAN-2013	1100	4100

Если до сих пор не все ясно, просто удалите агрегацию и начните с создания декартова произведения, используя таблицу `EMP`, чтобы можно было сравнить каждое значение `HIREDATE` со всеми другими значениями `HIREDATE`. Далее приводятся соответствующий запрос и небольшая часть его результирующего множества (полное декартово произведение таблицы `EMP` содержит $14 \times 14 = 196$ строк):

```
select e.hiredate,
       e.sal,
       d.sal,
       d.hiredate
  from emp e, emp d
```

HIREDATE	SAL	SAL	HIREDATE
17-DEC-2010	800	800	17-DEC-2010
17-DEC-2010	800	1600	20-FEB-2011

17-DEC-2010	800	1250	22-FEB-2011
17-DEC-2010	800	2975	02-APR-2011
17-DEC-2010	800	1250	28-SEP-2011
17-DEC-2010	800	2850	01-MAY-2011
17-DEC-2010	800	2450	09-JUN-2011
17-DEC-2010	800	3000	09-DEC-2012
17-DEC-2010	800	5000	17-NOV-2011
17-DEC-2010	800	1500	08-SEP-2011
17-DEC-2010	800	1100	12-JAN-2013
17-DEC-2010	800	950	03-DEC-2011
17-DEC-2010	800	3000	03-DEC-2011
17-DEC-2010	800	1300	23-JAN-2012
20-FEB-2011	1600	800	17-DEC-2010
20-FEB-2011	1600	1600	20-FEB-2011
20-FEB-2011	1600	1250	22-FEB-2011
20-FEB-2011	1600	2975	02-APR-2011
20-FEB-2011	1600	1250	28-SEP-2011
20-FEB-2011	1600	2850	01-MAY-2011
20-FEB-2011	1600	2450	09-JUN-2011
20-FEB-2011	1600	3000	09-DEC-2012
20-FEB-2011	1600	5000	17-NOV-2011
20-FEB-2011	1600	1500	08-SEP-2011
20-FEB-2011	1600	1100	12-JAN-2013
20-FEB-2011	1600	950	03-DEC-2011
20-FEB-2011	1600	3000	03-DEC-2011
20-FEB-2011	1600	1300	23-JAN-2012

Как можно видеть, полученное результирующее множество не содержит даты `HIREDATE`, равной или на 90 дней меньшей, чем 17 декабря, за исключением собственно 17 декабря. Поэтому сумма для этой строки должна быть 800. Для следующей даты `HIREDATE` — 20 февраля — есть только одна другая дата `HIREDATE`, входящая в 90-дневное окно (в пределах 90 дней до этой даты), — 17 декабря. Сложив `SAL` за 17 декабря и `SAL` за 20 февраля (поскольку мы ищем значения `HIREDATE`, равные текущей дате или в пределах 90 дней до нее), получим 2400, что будет конечным результатом для этой даты.

Разобравшись с этим процессом, используем фильтр в операторе `WHERE`, чтобы вернуть `SAL` для каждой даты `HIREDATE` и всех дат `HIREDATE`, равной ей или в пределах 90 дней до нее:

```
select e.hiredate,
       e.sal,
       d.sal sal_to_sum,
       d.hiredate within_90_days
from emp e, emp d
where d.hiredate
      between e.hiredate-90 and e.hiredate
order by 1
```

HIREDATE	SAL	SAL_TO_SUM	WITHIN_90_DAYS
17-DEC-2010	800	800	17-DEC-2010
20-FEB-2011	1600	800	17-DEC-2010
20-FEB-2011	1600	1600	20-FEB-2011
22-FEB-2011	1250	800	17-DEC-2010
22-FEB-2011	1250	1600	20-FEB-2011
22-FEB-2011	1250	1250	22-FEB-2011
02-APR-2011	2975	1600	20-FEB-2011
02-APR-2011	2975	1250	22-FEB-2011
02-APR-2011	2975	2975	02-APR-2011
01-MAY-2011	2850	1600	20-FEB-2011
01-MAY-2011	2850	1250	22-FEB-2011
01-MAY-2011	2850	2975	02-APR-2011
01-MAY-2011	2850	2850	01-MAY-2011
09-JUN-2011	2450	2975	02-APR-2011
09-JUN-2011	2450	2850	01-MAY-2011
09-JUN-2011	2450	2450	09-JUN-2011
08-SEP-2011	1500	1500	08-SEP-2011
28-SEP-2011	1250	1500	08-SEP-2011
28-SEP-2011	1250	1250	28-SEP-2011
17-NOV-2011	5000	1500	08-SEP-2011
17-NOV-2011	5000	1250	28-SEP-2011
17-NOV-2011	5000	5000	17-NOV-2011
03-DEC-2011	950	1500	08-SEP-2011
03-DEC-2011	950	1250	28-SEP-2011
03-DEC-2011	950	5000	17-NOV-2011
03-DEC-2011	950	950	03-DEC-2011
03-DEC-2011	950	3000	03-DEC-2011
03-DEC-2011	3000	1500	08-SEP-2011
03-DEC-2011	3000	1250	28-SEP-2011
03-DEC-2011	3000	5000	17-NOV-2011
03-DEC-2011	3000	950	03-DEC-2011
03-DEC-2011	3000	3000	03-DEC-2011
23-JAN-2012	1300	5000	17-NOV-2011
23-JAN-2012	1300	950	03-DEC-2011
23-JAN-2012	1300	3000	03-DEC-2011
23-JAN-2012	1300	1300	23-JAN-2012
09-DEC-2012	3000	3000	09-DEC-2012
12-JAN-2013	1100	3000	09-DEC-2012
12-JAN-2013	1100	1100	12-JAN-2013

Определив, какие значения SAL нужно включить в скользящее окно суммирования, просто используем агрегатную функцию SUM, чтобы создать более информативное результирующее множество:

```
select e.hiredate,
       e.sal,
       sum(d.sal) as spending_pattern
from emp e, emp d
```

```
where d.hiredate
      between e.hiredate-90 and e.hiredate
group by e.hiredate,e.sal
order by 1
```

Сравнив результирующее множество из предыдущего запроса с результирующим множеством следующего запроса (который является исходным решением), можно видеть, что они одинаковые:

```
select e.hiredate,
       e.sal,
       (select sum(sal) from emp d
        where d.hiredate between e.hiredate-90
                  and e.hiredate) as spending_pattern
from emp e
order by 1
```

HIREDATE	SAL	SPENDING_PATTERN
17-DEC-2010	800	800
20-FEB-2011	1600	2400
22-FEB-2011	1250	3650
02-APR-2011	2975	5825
01-MAY-2011	2850	8675
09-JUN-2011	2450	8275
08-SEP-2011	1500	1500
28-SEP-2011	1250	2750
17-NOV-2011	5000	7750
03-DEC-2011	950	11700
03-DEC-2011	3000	11700
23-JAN-2012	1300	10250
09-DEC-2012	3000	3000
12-JAN-2013	1100	4100

12.20. Транспонирование результирующего множества, содержащего подсуммы

ЗАДАЧА

Требуется создать отчет, содержащий подсуммы, а затем транспонировать полученное результирующее множество, чтобы сделать отчет более удобочитаемым. Например, надо создать отчет, содержащий менеджеров для каждого отдела и суммы зарплат служащих, работающих в подчинении этих менеджеров. Кроме этого, надо также возвратить две подсуммы: сумму всех зарплат по отделам для служащих, работающих под руководством какого-либо менеджера, и сумму всех зарплат в результирующем множестве (сумму всех подсумм по отделам). Исходный отчет выглядит следующим образом:

DEPTNO	MGR	SAL
10	7782	1300
10	7839	2450
10		3750
20	7566	6000
20	7788	1100
20	7839	2975
20	7902	800
20		10875
30	7698	6550
30	7839	2850
30		9400
		24025

Этот отчет нужно сделать более удобочитаемым, преобразовав его в следующий, более ясно отображающий значение предоставленных данных:

MGR	DEPT10	DEPT20	DEPT30	TOTAL
7566	0	6000	0	
7698	0	0	6550	
7782	1300	0	0	
7788	0	1100	0	
7839	2450	2975	2850	
7902	0	800	0	
	3750	10875	9400	24025

РЕШЕНИЕ

На первом шаге формируем подсуммы с помощью расширения `ROLLUP` оператора `GROUP BY`. Затем выполняем классическое транспонирование (используя агрегатную функцию и выражение `CASE`), чтобы создать требуемые столбцы для нужного отчета. Функция `GROUPING` позволяет с легкостью определить значения, являющиеся подсуммами (т. е. созданные расширением `ROLLUP` и отсутствовавшие бы в противном случае). В зависимости от подхода используемой СУБД к сортировке значений `NULL`, может потребоваться добавить в решение оператор `ORDER BY`, чтобы полученное результирующее множество соответствовало требуемому, показанному в *разд. «Задача»*.

DB2 и Oracle

Для представления данных в более удобочитаемом формате сначала используем расширение `ROLLUP` оператора `GROUP BY`, а затем выражение `CASE`:

```

1 select mgr,
2     sum(case deptno when 10 then sal else 0 end) dept10,
3     sum(case deptno when 20 then sal else 0 end) dept20,
4     sum(case deptno when 30 then sal else 0 end) dept30,
5     sum(case flag when '11' then sal else null end) total

```

```

6   from (
7 select deptno,mgr,sum(sal) sal,
8       cast(grouping(deptno) as char(1))||
9       cast(grouping(mgr) as char(1)) flag
10  from emp
11  where mgr is not null
12  group by rollup(deptno,mgr)
13         ) x
14  group by mgr

```

SQL Server

Для представления данных в более удобочитаемом формате сначала используем расширение ROLLUP оператора GROUP BY, а затем выражение CASE:

```

1 select mgr,
2       sum(case deptno when 10 then sal else 0 end) dept10,
3       sum(case deptno when 20 then sal else 0 end) dept20,
4       sum(case deptno when 30 then sal else 0 end) dept30,
5       sum(case flag when '11' then sal else null end) total
6   from (
7 select deptno,mgr,sum(sal) sal,
8       cast(grouping(deptno) as char(1))+
9       cast(grouping(mgr) as char(1)) flag
10  from emp
11  where mgr is not null
12  group by deptno,mgr with rollup
13         ) x
14  group by mgr

```

PostgreSQL

Для представления данных в более удобочитаемом формате сначала используем расширение ROLLUP оператора GROUP BY, а затем выражение CASE:

```

1 select mgr,
2       sum(case deptno when 10 then sal else 0 end) dept10,
3       sum(case deptno when 20 then sal else 0 end) dept20,
4       sum(case deptno when 30 then sal else 0 end) dept30,
5       sum(case flag when '11' then sal else null end) total
6   from (
7 select deptno,mgr,sum(sal) sal,
8       concat(cast (grouping(deptno) as char(1)),
9             cast(grouping(mgr) as char(1))) flag
10  from emp
11  where mgr is not null
12  group by rollup (deptno,mgr)
13         ) x
14  group by mgr

```


MySQL

Для представления данных в более удобочитаемом формате сначала используем расширение `ROLLUP` оператора `GROUP BY`, а затем выражение `CASE`:

```

1 select mgr,
2     sum(case deptno when 10 then sal else 0 end) dept10,
3     sum(case deptno when 20 then sal else 0 end) dept20,
4     sum(case deptno when 30 then sal else 0 end) dept30,
5     sum(case flag when '11' then sal else null end) total
6   from (
7 select deptno,mgr,sum(sal) sal,
8     concat( cast(grouping(deptno) as char(1)) ,
9     cast(grouping(mgr) as char(1))) flag
10  from emp
11  where mgr is not null
12  group by deptno,mgr with rollup
13         ) x
14  group by mgr;
```

Обсуждение

Представленные решения практически идентичны, за исключением оформления конкатенации строк и задания группирования `GROUPING`. Поэтому для обсуждения промежуточных результатов мы будем ссылаться на решение для SQL Server (но все сказанное также применимо и к DB2, и к Oracle).

На первом шаге создаем результирующее множество, в котором суммируются зарплаты `SAL` всех служащих каждого менеджера `MGR` для каждого отдела `DEPTNO`. Цель состоит в том, чтобы показать, сколько зарабатывают служащие под руководством определенного менеджера в определенном отделе. Например, следующий запрос позволит нам сравнить зарплаты служащих, работающих под началом менеджера `KING` в отделе `DEPTNO 10` с зарплатами служащих, работающих под началом этого же менеджера в отделе `DEPTNO 30`:

```

select deptno,mgr,sum(sal) sal
  from emp
  where mgr is not null
  group by mgr,deptno
  order by 1,2
```

DEPTNO	MGR	SAL
10	7782	1300
10	7839	2450
20	7566	6000
20	7788	1100
20	7839	2975
20	7902	800
30	7698	6550
30	7839	2850

Затем с помощью расширения `ROLLUP` оператора `GROUP BY` вычисляем подсуммы зарплат для каждого отдела `DEPTNO` и для всех служащих (находящихся в чем-либо подчинении):

```
select deptno,mgr,sum(sal) sal
       from emp
       where mgr is not null
       group by deptno,mgr with rollup
```

DEPTNO	MGR	SAL
10	7782	1300
10	7839	2450
10		3750
20	7566	6000
20	7788	1100
20	7839	2975
20	7902	800
20		10875
30	7698	6550
30	7839	2850
30		9400
		24025

Создав подсуммы, нужно определить, какие из этих значений действительно являются подсуммами (созданными расширением `ROLLUP`), а какие были созданы обычным оператором `GROUP BY`. Эта задача решается созданием битовых карт с помощью функции `GROUPING`:

```
select deptno,mgr,sum(sal) sal,
       cast(grouping(deptno) as char(1))+
       cast(grouping(mgr) as char(1)) flag
       from emp
       where mgr is not null
       group by deptno,mgr with rollup
```

DEPTNO	MGR	SAL	FLAG
10	7782	1300	00
10	7839	2450	00
10	3750		01
20	7566	6000	00
20	7788	1100	00
20	7839	2975	00
20	7902	800	00
20	10875		01
30	7698	6550	00
30	7839	2850	00
30	9400		01
		24025	11

Строки со значениями 00 столбца FLAG созданы обычными операциями агрегации. Строки же, для которых значение этого столбца равно 01, являются результатом агрегации значений SAL расширением ROLLUP по DEPTNO, т. к. DEPTNO указан в списке ROLLUP первым. Если изменить порядок, например, так:

```
GROUP BY MGR, DEPTNO WITH ROLLUP
```

результаты будут существенно иными. Наконец, строка со значением столбца FLAG, равным 11, создана в результате суммирования расширением ROLLUP всех значений SAL.

На этом этапе мы располагаем всем необходимым для создания облагороженного отчета, просто используя выражения CASE. Наша цель — создать отчет, отображающий зарплаты служащих всех менеджеров по отделам. Если у менеджера нет подчиненных в определенном отделе, возвращается ноль, в противном случае нужно вернуть сумму всех зарплат подчиненных этого менеджера в этом отделе. Кроме этого, нужно добавить столбец TOTAL, содержащий сумму всех зарплат в отчете. Далее приводится запрос, удовлетворяющий всем этим условиям, и результаты его исполнения:

```
select mgr,
       sum(case deptno when 10 then sal else 0 end) dept10,
       sum(case deptno when 20 then sal else 0 end) dept20,
       sum(case deptno when 30 then sal else 0 end) dept30,
       sum(case flag when '11' then sal else null end) total
  from (
select deptno,mgr,sum(sal) sal,
       cast(grouping(deptno) as char(1))+
       cast(grouping(mgr) as char(1)) flag
  from emp
 where mgr is not null
 group by deptno,mgr with rollup
        ) x
 group by mgr
 order by coalesce(mgr,9999)
```

MGR	DEPT10	DEPT20	DEPT30	TOTAL
7566	0	6000	0	
7698	0	0	6550	
7782	1300	0	0	
7788	0	1100	0	
7839	2450	2975	2850	
7902	0	800	0	
	3750	10875	9400	24025

12.21. Подведем итоги

Базы данных предназначены для хранения данных, но в конечном итоге кому-либо нужно извлечь эти данные, чтобы представить их где-либо. Рецепты этой главы демонстрируют разные важные способы переформатирования данных, чтобы удовлетворить требования пользователей. Кроме общей полезности предоставления пользователям данных в требуемой форме, эти методы играют важную роль в предоставлении владельцу базы данных возможности для создания удобного хранилища данных.

По мере приобретения опыта в области предоставления поддержки бизнес-пользователям вы станете более компетентными в использовании содержащихся здесь советов и сможете расширить их для создания более сложных презентаций.

Иерархические запросы

В этой главе рассматриваются способы выражения иерархических отношений, которые могут существовать в данных. Извлечение и отображение иерархических данных (в иерархическом виде) обычно представляет больше трудностей, чем их сохранение.

В настоящее время, когда пару лет назад рекурсивные обобщенные табличные выражения были добавлены и в MySQL, их поддержка присутствует практически во всех СУБД. В результате они являются эталоном для работы с иерархическими запросами, и в этой главе мы широко используем предоставляемые ими возможности для создания рецептов, описывающих работу с иерархически структурированными данными.

Но прежде чем приступить к рассмотрению таких рецептов, обратимся к таблице EMP и исследуем иерархические взаимосвязи между данными EMPNO и MGR. Далее приводится соответствующий запрос и его результаты:

```
select empno,mgr
       from emp
       order by 2
```

EMPNO	MGR
7788	7566
7902	7566
7499	7698
7521	7698
7900	7698
7844	7698
7654	7698
7934	7782
7876	7788
7566	7839
7782	7839
7698	7839
7369	7902
7839	

При внимательном рассмотрении полученного результирующего множества можно увидеть, что каждое значение столбца MGR также присутствует и в столбце EMPNO. Это означает, что руководитель каждого служащего в таблице EMP также является и

служащим, информация о котором тоже хранится в таблице EMP, а не где-либо в ином месте. Взаимосвязь между MGR и EMPNO — это отношение типа родитель-потомок, т. к. значение MGR является прямым родителем значений EMPNO. (Также возможно, что руководитель находится в подчинении своего руководителя, который, в свою очередь, может работать еще под чьим-либо руководством, и т. д., в результате чего создается иерархия в n уровней.) Если у служащего нет руководителя, то соответствующее поле MGR имеет значение NULL.

13.1. Выражение отношений родитель—потомок

ЗАДАЧА

Требуется вместе с данными записи потомков также включить информацию их родителей. Например, надо вывести имя каждого служащего вместе с именем его руководителя. Результирующее множество должно иметь следующий вид:

```
EMPS_AND_MGRS
-----
FORD works for JONES
SCOTT works for JONES
JAMES works for BLAKE
TURNER works for BLAKE
MARTIN works for BLAKE
WARD works for BLAKE
ALLEN works for BLAKE
MILLER works for CLARK
ADAMS works for SCOTT
CLARK works for KING
BLAKE works for KING
JONES works for KING
SMITH works for FORD
```

РЕШЕНИЕ

Чтобы определить имя руководителя каждого служащего, сначала выполняем самообъединение таблицы EMP по столбцам MGR и EMPNO. Затем применяем функции для конкатенирования строк, поддерживаемые используемой СУБД, чтобы сформировать строки в требуемое результирующее множество.

DB2, Oracle и PostgreSQL

Сначала выполняем самообъединение таблицы EMP, а затем конкатенируем строки в требуемое результирующее множество с помощью оператора конкатенации ||:

```
1 select a.ename || ' works for ' || b.ename as emps_and_mgrs
2     from emp a, emp b
3     where a.mgr = b.empno
```

MySQL

Сначала выполняем самообъединение таблицы EMP, а затем конкатенируем строки в требуемое результирующее множество с помощью функции конкатенации `CONCAT`:

```
1 select concat(a.ename, ' works for ',b.ename) as emps_and_mgrs
2     from emp a, emp b
3     where a.mgr = b.empno
```

SQL Server

Сначала выполняем самообъединение таблицы EMP, а затем конкатенируем строки в требуемое результирующее множество с помощью оператора конкатенации `+`:

```
1 select a.ename + ' works for ' + b.ename as emps_and_mgrs
2     from emp a, emp b
3     where a.mgr = b.empno
```

Обсуждение

Реализация решения практически одинакова для всех СУБД. Они различаются между собой только способами конкатенации строк, поэтому это обсуждение применимо ко всем решениям.

Ключевой момент решения — объединение по столбцам `MGR` и `EMPNO`. На первом шаге выполняем самообъединение таблицы EMP, чтобы получить декартово произведение. Далее показан соответствующий запрос и часть возвращаемых им строк:

```
select a.empno, b.empno mgr
       from emp a, emp b
```

EMPNO	EMPNO
7369	7369
7369	7499
7369	7521
7369	7566
7369	7654
7369	7698
7369	7782
7369	7788
7369	7839
7369	7844
7369	7876
7369	7900
7369	7902
7369	7934
7499	7369
7499	7499
7499	7521
7499	7566

```

7499      7654
7499      7698
7499      7782
7499      7788
7499      7839
7499      7844
7499      7876
7499      7900
7499      7902
7499      7934

```

Как можно видеть, декартово произведение возвращает все возможные комбинации EMPNO/EMPNO (как будто руководителем служащего, например EMPNO 7369, являются все другие служащие в таблице, включая самого служащего 7369).

Затем отфильтровываем результаты таким образом, чтобы вернуть только каждого служащего и номер EMPNO его руководителя. Для этого выполняем самообъединение таблицы EMP по столбцам EMPNO и MGR. Далее показан соответствующий запрос и часть его результирующего множества:

```

1 select a.empno, b.empno mgr
2    from emp a, emp b
3   where a.mgr = b.empno

```

EMPNO	MGR
7902	7566
7788	7566
7900	7698
7844	7698
7654	7698
7521	7698
7499	7698
7934	7782
7876	7788
7782	7839
7698	7839
7566	7839
7369	7902

Теперь, когда у нас есть номер EMPNO каждого служащего и их руководителей, мы можем вернуть имена всех руководителей, просто указав в запросе B.ENAME вместо B.EMPNO. В случае трудностей с пониманием этого процесса для получения ответа вместо самообъединения можно использовать скалярный подзапрос:

```

select a.ename,
       (select b.ename
        from emp b
        where b.empno = a.mgr) as mgr
from emp a

```


ENAME	MGR
SMITH	FORD
ALLEN	BLAKE
WARD	BLAKE
JONES	KING
MARTIN	BLAKE
BLAKE	KING
CLARK	KING
SCOTT	JONES
KING	
TURNER	BLAKE
ADAMS	SCOTT
JAMES	BLAKE
FORD	JONES
MILLER	CLARK

Результирующее множество скалярной версии решения эквивалентно множеству версии с использованием самообъединения, за исключением одной строки — оно содержит служащего KING, которого нет во множестве с использованием самообъединения. Можно спросить: почему так? Вспомним, что значение NULL никогда не равно никакому значению, даже другому значению NULL. В решении с самообъединением выполняется эквиобъединение по столбцам EMPNO и MGR, в результате чего отфильтровываются все служащие, для которых значение MGR равно NULL. В решении с самообъединением для включения в результирующее множество служащего KING самообъединение нужно выполнить, как показано в следующих двух запросах. В первом из них внешнее самообъединение выполняется по стандарту ANSI, а во втором используется синтаксис Oracle. Оба решения создают одно и то же результирующее множество:

```
/* ANSI */
select a.ename, b.ename mgr
       from emp a left join emp b
          on (a.mgr = b.empno)
```

```
/* Oracle */
select a.ename, b.ename mgr
       from emp a, emp b
       where a.mgr = b.empno (+)
```

ENAME	MGR
FORD	JONES
SCOTT	JONES
JAMES	BLAKE
TURNER	BLAKE
MARTIN	BLAKE

```

WARD      BLAKE
ALLEN     BLAKE
MILLER    CLARK
ADAMS     SCOTT
CLARK     KING
BLAKE     KING
JONES     KING
SMITH     FORD
KING

```

13.2. Выражение отношений потомок—родитель—прародитель

ЗАДАЧА

Служащий CLARK работает в подчинении руководителя KING. Это отношение можно выразить с помощью первого рецепта этой главы. Но что, если CLARK, в свою очередь, является руководителем другого служащего? Рассмотрим следующий запрос:

```

select ename,empno,mgr
       from emp
       where ename in ('KING','CLARK','MILLER')

```

ENAME	EMPNO	MGR
CLARK	7782	7839
KING	7839	
MILLER	7934	7782

Как можно видеть, служащий MILLER работает в подчинении у CLARK, который, в свою очередь, работает под руководством KING. Нам нужно выразить всю иерархию от MILLER до KING, чтобы получить следующее результирующее множество:

```

LEAF__BRANCH__ROOT
-----
MILLER-->CLARK-->KING

```

Разумеется, для полного представления отношений сверху донизу одного самообъединения, показанного в предыдущем рецепте, недостаточно. Задачу можно было бы решить с помощью запроса, выполняющего два самообъединения, но что нам в действительности нужно, так это найти общий метод для обхода таких иерархий.

РЕШЕНИЕ

Этот рецепт отличается от предыдущего присутствием трехуровневого отношения, что и отражено в его названии. Если используемая СУБД не предоставляет функциональность для обхода данных с древовидной структурой, как в случае с Oracle, эту задачу можно решить с помощью обобщенных табличных выражений.

DB2, PostgreSQL и SQL Server

Чтобы определить руководителя служащего MILLER, которым является CLARK, а затем руководителя служащего CLARK, которым является KING, используем рекурсивный оператор WITH. В этом решении также применяется оператор конкатенации строк SQL Server +:

```

1  with x (tree,mgr,depth)
2  as (
3  select cast(ename as varchar(100)),
4         mgr, 0
5  from emp
6  where ename = 'MILLER'
7  union all
8  select cast(x.tree+'-->' + e.ename as varchar(100)),
9         e.mgr, x.depth+1
10 from emp e, x
11 where x.mgr = e.empno
12 )
13 select tree leaf__branch__root
14 from x
15 where depth = 2

```

Это решение применимо к DB2 и PostgreSQL при условии использования соответствующего оператора конкатенации. В частности, для DB2 используется оператор ||, а для PostgreSQL — оператор CONCAT.

MySQL

Это решение подобно предыдущему, но в нем нужно использовать ключевое слово RECURSIVE:

```

1  with recursive x (tree,mgr,depth)
2  as (
3  select cast(ename as varchar(100)),
4         mgr, 0
5  from emp
6  where ename = 'MILLER'
7  union all
8  select cast(concat(x.tree,'-->',emp.ename) as char(100)),
9         e.mgr, x.depth+1
10 from emp e, x
11 where x.mgr = e.empno
12 )
13 select tree leaf__branch__root
14 from x
15 where depth = 2

```

Oracle

Чтобы определить руководителя служащего MILLER, которым является CLARK, а затем руководителя служащего CLARK, которым является KING, используем функцию SYS_CONNECT_BY_PATH. Для обхода дерева используем оператор CONNECT BY:

```
1 select ltrim(
2 sys_connect_by_path(ename, '-->'),
3 '-->') leaf branch root
4 from emp
5 where level = 3
6 start with ename = 'MILLER'
7 connect by prior mgr = empno
```

Обсуждение

DB2, SQL Server, PostgreSQL и MySQL

Подход к решению этой задачи — начать обход дерева с концевой узла и продолжать по направлению к корневому. (Весьма полезно было бы также попробовать выполнить обход в противоположном направлении.) Верхняя часть оператора UNION ALL просто находит строку служащего MILLER (концевой узел). А нижняя часть этого оператора находит служащего, который является руководителем MILLER, а затем руководителя этого служащего, являющегося одновременно и руководителем. Такой процесс нахождения старшего руководителя младшего руководителя повторяется до тех пор, пока не достигнет руководителя наивысшего уровня (корневой узел). Значение DEPTH начинается с 0 и автоматически инкрементируется на 1 при каждом нахождении руководителя (DEPTH — это значение, которое обрабатывается DB2 при исполнении рекурсивных запросов).



Интересное и всестороннее рассмотрение оператора WITH с упором на его рекурсивное применение предоставляется в статье Джонатана Генника (Jonathan Gennick) «Understanding the WITH Clause» («Понимание оператора WITH»), которую можно найти по адресу <http://gennick.com/with.htm>.

Затем второй запрос оператора UNION ALL объединяет рекурсивное представление X с таблицей EMP, чтобы определить отношение родитель—потомок. На этом этапе запрос (с использованием оператора конкатенирования SQL Server) выглядит следующим образом:

```
with x (tree,mgr,depth)
  as (
select cast(ename as varchar(100)),
      mgr, 0
  from emp
  where ename = 'MILLER'
  union all
select cast(x.tree+'-->'||e.ename as varchar(100)),
      e.mgr, x.depth+1
  from emp e, x
```

```

where x.mgr = e.empno
)
select tree leaf__branch__root
       from x

```

```

TREE          DEPTH
-----
MILLER                0
CLARK                 1
KING                  2

```

К этому моменту основная часть задачи решена — начиная со служащего MILLER, возвращены все иерархические отношения снизу доверху. Осталось только отформатировать результаты. Поскольку обход дерева выполняется рекурсивно, просто конкатенируем в следующем запросе текущее значение ENAME таблицы EMP с предшествующим значением, что даст нам такое результирующее множество:

```

with x (tree,mgr,depth)
  as (
select cast(ename as varchar(100)),
       mgr, 0
  from emp
 where ename = 'MILLER'
 union all
select cast(x.tree+'-->'||e.ename as varchar(100)),
       e.mgr, x.depth+1
  from emp e, x
 where x.mgr = e.empno
)
select depth, tree
       from x

```

```

DEPTH TREE
-----
0 MILLER
1 MILLER-->CLARK
2 MILLER-->CLARK-->KING

```

В завершение выбираем только последнюю строку результирующего множества иерархии. Эту задачу можно решить несколькими способами. В решении для определения достижения корневого узла используется значение DEPTH. Очевидно, что, если бы руководителем CLARK был бы кто-то другой, а не KING, фильтр по DEPTH нужно было бы изменить. Более общее решение, не требующее такого фильтра, рассматривается в следующем рецепте.

Oracle

В решении для Oracle вся работа выполняется оператором CONNECT BY. Начиная с MILLER, обходим все дерево до KING без необходимости выполнять какие бы то ни

было объединения. Выражение в операторе `CONNECT BY` определяет взаимосвязь данных и метод обхода дерева:

```
select ename
       from emp
       start with ename = 'MILLER'
       connect by prior mgr = empno
```

```
ENAME
-----
MILLER
CLARK
KING
```

Ключевое слово `PRIOR` позволяет обращаться к значениям из предыдущей записи иерархии. То есть обращаться к номеру руководителя служащего с любым `EMPNO` можно посредством `PRIOR MGR`. Конструкцию наподобие

```
CONNECT BY PRIOR MGR = EMPNO
```

можно рассматривать как выражение объединения между — в нашем случае — родителем и потомком.



Более подробную информацию по оператору Oracle `CONNECT BY` и его применению в иерархических запросах можно найти в статье «Hierarchical Queries in Oracle» («Иерархические запросы в Oracle») по адресу: <https://oreil.ly/6yfh>.

На этом этапе мы успешно отобразили всю иерархию, начиная с `MILLER` и заканчивая `KING`. Задача по большому счету решена — осталось только выполнить форматирование. Для этого воспользуемся функцией `SYS_CONNECT_BY_PATH`, которая добавляет каждое значение `ENAME` к предыдущему значению:

```
select sys_connect_by_path(ename, '-->') tree
       from emp
       start with ename = 'MILLER'
       connect by prior mgr = empno
```

```
TREE
-----
-->MILLER
-->MILLER-->CLARK
-->MILLER-->CLARK-->KING
```

Так как нам нужна только полная иерархия, отфильтровываем неполные строки по псевдостолбцу `LEVEL` (более общий подход рассматривается в следующем рецепте):

```
select sys_connect_by_path(ename, '-->') tree
       from emp
       where level = 3
       start with ename = 'MILLER'
       connect by prior mgr = empno
```

```
TREE
-----
-->MILLER-->CLARK-->KING
```

В завершение с помощью функции `LTRIM` удаляем из результирующего множества ведущие символы `-->`.

13.3. Создание иерархического представления таблицы

ЗАДАЧА

Требуется вернуть результирующее множество, описывающее иерархию всей таблицы. В случае таблицы `EMP` у служащего `KING` нет руководителя, поэтому он представляет собой корневой узел. Требуется отобразить, начиная со служащего `KING`, всех служащих под его руководством, а также всех служащих (если таковые имеются), руководителями которых являются служащие, подчиненные руководителю `KING`. Конечное результирующее множество должно иметь следующий вид:

```
EMP_TREE
-----
KING
KING - BLAKE
KING - BLAKE - ALLEN
KING - BLAKE - JAMES
KING - BLAKE - MARTIN
KING - BLAKE - TURNER
KING - BLAKE - WARD
KING - CLARK
KING - CLARK - MILLER
KING - JONES
KING - JONES - FORD
KING - JONES - FORD - SMITH
KING - JONES - SCOTT
KING - JONES - SCOTT - ADAMS
```

РЕШЕНИЕ

DB2, PostgreSQL и SQL Server

Для построения иерархии, начинающейся со служащего `KING` и в конечном итоге отображающей всех служащих, используем рекурсивный оператор `WITH`. В представленном далее решении используется оператор конкатенации `DB2` — символ `||`. В `SQL Server` в качестве оператора конкатенации применяется символ `+`, а для `PostgreSQL` — функция `CONCAT`. С поправкой на соответствующий оператор или функцию конкатенации решение в представленном виде будет также работать и в этих двух СУБД:

```

1  with x (ename,empno)
2    as (
3  select cast(ename as varchar(100)),empno
4    from emp
5   where mgr is null
6   union all
7  select cast(x.ename||' - '||e.ename as varchar(100)),
8         e.empno
9    from emp e, x
10   where e.mgr = x.empno
11 )
12 select ename as emp_tree
13    from x
14   order by 1

```

MySQL

В MySQL нужно еще использовать ключевое слово RECURSIVE:

```

1  with recursive x (ename,empno)
2    as (
3  select cast(ename as varchar(100)),empno
4    from emp
5   where mgr is null
6   union all
7  select cast(concat(x.ename,' - ',e.ename) as varchar(100)),
8         e.empno
9    from emp e, x
10   where e.mgr = x.empno
11 )
12 select ename as emp_tree
13    from x
14   order by 1

```

Oracle

Для определения иерархии используем функцию CONNECT BY, а затем форматируем вывод требуемым образом с помощью функции SYS_CONNECT_BY_PATH:

```

1  select ltrim(
2         sys_connect_by_path(ename,' - '),
3         ' - ') emp_tree
4    from emp
5   start with mgr is null
6  connect by prior empno=mgr
7   order by 1

```

Это решение отличается от решения предыдущего рецепта отсутствием фильтра по псевдостолбцу LEVEL. Без такого фильтра отображаются все возможные деревья (где PRIOR EMPNO=MGR).

Обсуждение

DB2, MySQL, PostgreSQL и SQL Server

На первом шаге в верхней части конструкции UNION ALL в рекурсивном запросе X определяем корневую строку (это служащий KING). Затем определяем подчиненных KING и подчиненных этих служащих, если таковые имеются, выполнив объединение рекурсивного представления X с таблицей EMP. Рекурсия будет продолжаться до тех пор, пока не будут возвращены все служащие. Далее приводится соответствующий запрос и возвращаемое им не отформатированное результирующее множество:

```
with x (ename,empno)
  as (
select cast(ename as varchar(100)),empno
  from emp
  where mgr is null
  union all
select cast(e.ename as varchar(100)),e.empno
  from emp e, x
  where e.mgr = x.empno
)
select ename emp_tree
  from x
```

EMP_TREE

KING

JONES

SCOTT

ADAMS

FORD

SMITH

BLAKE

ALLEN

WARD

MARTIN

TURNER

JAMES

CLARK

MILLER

Возвращены все строки иерархии, но без надлежащего форматирования нельзя определить руководителей. Чтобы вернуть более значащий результат, нужно объединить всех служащих с их соответствующими руководителями. Для этого просто используем в рекурсивном запросе X следующее выражение в операторе SELECT нижней части конструкции UNION ALL:

```
cast(x.ename+', '+e.ename as varchar(100))
```

Оператор `WITH` чрезвычайно полезен для решения подобных задач, поскольку позволяет не изменять запрос при изменении иерархии (например, когда концевые узлы превращаются в узлы ветвления).

Oracle

Оператор `CONNECT BY` возвращает строки иерархии, а оператор `START WITH` определяет корневой узел. Исполнив решение без функции `SYS_CONNECT_BY_PATH`, можно убедиться в том, что возвращаются правильные строки (что может быть полезным при отладке), но без форматирования, выражающего взаимосвязь между ними:

```
select ename emp_tree
       from emp
       start with mgr is null
       connect by prior empno = mgr
```

```
EMP_TREE
-----
KING
JONES
SCOTT
ADAMS
FORD
SMITH
BLAKE
ALLEN
WARD
MARTIN
TURNER
JAMES
CLARK
MILLER
```

Оформить результат для более ясного представления иерархии можно с помощью псевдостолбца `LEVEL` и функции `LPAD` и в конечном счете понять, почему функция `SYS_CONNECT_BY_PATH` возвращает требуемый результат, показанный ранее:

```
select lpad(' ', 2*level, '.')||ename emp_tree
       from emp
       start with mgr is null
       connect by prior empno = mgr
```

```
EMP_TREE
-----
..KING
...JONES
....SCOTT
.....ADAMS
.....FORD
.....SMITH
```

```

...BLAKE
.....ALLEN
.....WARD
.....MARTIN
.....TURNER
.....JAMES
....CLARK
.....MILLER

```

Иерархия в результирующем множестве выражается бóльшими отступами для подчиненных, расположенных под руководителем с меньшим отступом. Например, служащий KING не находится ни в чем подчинении, JONES находится в подчинении KING, SCOTT — в подчинении JONES, а ADAMS — в подчинении SCOTT.

Если посмотреть на соответствующие строки решения с использованием функции SYS_CONNECT_BY_PASS, можно увидеть, что эта функция сворачивает иерархию. В частности, при отображении нового узла также отображаются и все его предыдущие узлы:

```

KING
KING - JONES
KING - JONES - SCOTT
KING - JONES - SCOTT - ADAMS

```

13.4. Выборка всех дочерних строк заданной родительской строки

ЗАДАЧА

Требуется определить всех служащих, работающих под руководством JONES, прямым или косвенным (т. е. под руководством кого-либо находящегося в подчинении у JONES). Далее приводится список всех служащих, находящихся в подчинении JONES (включая самого JONES):

```

ENAME
-----
JONES
SCOTT
ADAMS
FORD
SMITH

```

РЕШЕНИЕ

Возможность доступа к самому верхнему или нижнему узлу дерева чрезвычайно полезна. Для этого решения не требуется выполнять никакого специального форматирования. Наша цель — просто вернуть всех служащих, работающих прямо или косвенно в подчинении JONES, включая самого JONES. Подобные задачи реально

демонстрируют полезность рекурсивных расширений SQL, таких как операторы `CONNECT BY` для Oracle и `WITH` для SQL Server и DB2.

DB2, PostgreSQL и SQL Server

Чтобы вернуть всех служащих под руководством JONES, используем рекурсивный оператор `WITH`. Обход дерева начинаем с JONES, указав конструкцию:

```
WHERE ENAME = 'JONES'
```

в первом из двух запросов объединения:

```
1  with x (ename,empno)
2    as (
3  select ename,empno
4    from emp
5   where ename = 'JONES'
6   union all
7  select e.ename, e.empno
8    from emp e, x
9   where x.empno = e.mgr
10 )
11 select ename
12    from x
```

Oracle

Чтобы найти всех служащих под руководством JONES, используем оператор `CONNECT BY`, задав при этом конструкцию `START WITH ENAME = 'JONES'`:

```
1 select ename
2    from emp
3   start with ename = 'JONES'
4  connect by prior empno = mgr
```

Обсуждение

DB2, MySQL, PostgreSQL и SQL Server

Благодаря рекурсивному оператору `WITH` эта задача легко поддается решению. Первая часть этого оператора (верхняя часть `UNION ALL`) возвращает строку для служащего JONES. Чтобы получить имя, нужно вернуть `ENAME`, а для выполнения объединения — вернуть `EMPNO`. В нижней части `UNION ALL` выполняется рекурсивное объединение `EMP.MGR` с `X.EMPNO`. Условие объединения применяется до тех пор, пока не завершится результирующее множество.

Oracle

Оператор `START WITH` указывает запросу сделать JONES корневым узлом. Условие в операторе `CONNECT BY` управляет обходом дерева, который исполняется до тех пор, пока условие истинно.

13.5. Определение концевых, неконцевых и корневых узлов

ЗАДАЧА

Требуется определить, узлом какого типа является заданная строка: концевым, неконцевым (узлом ветвления) или корневым. Примем для этого примера, что концевым узлом считается служащий, не являющийся руководителем. Узлом ветвления считается служащий, который является руководителем и одновременно находится в чем-либо подчинении. Корневым узлом считается служащий, не имеющий руководителя. Статус каждой строки в иерархии отображается возвращением значения 1 (Истина) или 0 (Ложь). Надо вернуть следующее результирующее множество:

ENAME	IS_LEAF	IS_BRANCH	IS_ROOT
KING	0	0	1
JONES	0	1	0
SCOTT	0	1	0
FORD	0	1	0
CLARK	0	1	0
BLAKE	0	1	0
ADAMS	1	0	0
MILLER	1	0	0
JAMES	1	0	0
TURNER	1	0	0
ALLEN	1	0	0
WARD	1	0	0
MARTIN	1	0	0
SMITH	1	0	0

РЕШЕНИЕ

Важно понимать, что таблица EMP моделируется по древовидной, а не по рекурсивной иерархии и что значение MGR корневых узлов равно NULL. Если бы таблица EMP моделировалась с использованием рекурсивной иерархии, ее корневые узлы ссылались бы сами на себя (т. е. значением MGR для служащего KING было бы его же значение EMPNO). Мы находим самоссылки трудными для понимания и поэтому для значений MGR корневых узлов используем значения NULL. Операторы CONNECT BY для Oracle и WITH для DB2 и SQL Server более удобны и, потенциально, более эффективны для обработки древовидных иерархий, чем рекурсивных иерархий. В случае обработки рекурсивной иерархии с помощью CONNECT BY или WITH следует быть внимательным, чтобы случайно не создать цикл в коде SQL. Поэтому при необходимости работать с рекурсивной иерархией код следует разрабатывать таким образом, чтобы не допустить подобного развития.

DB2, PostgreSQL, MySQL и SQL Server

Булево значение, которое нужно вернуть для каждого узла (1 или 0), определяем с помощью трех скалярных подзапросов:

```

1 select e.ename,
2     (select sign(count(*)) from emp d
3     where 0 =
4     (select count(*) from emp f
5     where f.mgr = e.empno)) as is_leaf,
6     (select sign(count(*)) from emp d
7     where d.mgr = e.empno
8     and e.mgr is not null) as is_branch,
9     (select sign(count(*)) from emp d
10    where d.empno = e.empno
11    and d.mgr is null) as is_root
12 from emp e
13 order by 4 desc,3 desc

```

Oracle

Подход с использованием скалярных подзапросов также применим и для Oracle. Более того, для версий Oracle, предшествующих Oracle Database 10g, этот подход единственно возможный. Однако в представленном далее решении для определения корневых и конечных строк используются встроенные функции Oracle, которые были введены в Oracle Database 10g. Это функции `CONNECT_BY_ROOT` и `CONNECT_BY_ISLEAF`, соответственно:

```

1 select ename,
2     connect_by_isleaf is_leaf,
3     (select count(*) from emp e
4     where e.mgr = emp.empno
5     and emp.mgr is not null
6     and rownum = 1) is_branch,
7     decode(ename,connect_by_root(ename),1,0) is_root
8 from emp
9 start with mgr is null
10 connect by prior empno = mgr
11 order by 4 desc, 3 desc

```

Обсуждение

DB2, PostgreSQL, MySQL и SQL Server

Для определения строк конечных, неконцевых и корневых узлов решение просто применяет правила, заданные в *разд. «Задача»*. На первом шаге определяем, является ли строка служащего конечным узлом. Концевыми узлами будут строки тех служащих, которые не являются руководителями, т. е. не имеют других служащих

в своем подчинении. Далее приводится соответствующий скалярный подзапрос IS_LEAF и его результаты:

```
select e.ename,
       (select sign(count(*)) from emp d
        where 0 =
          (select count(*) from emp f
           where f.mgr = e.empno)) as is_leaf
from emp e
order by 2 desc
```

ENAME	IS_LEAF
SMITH	1
ALLEN	1
WARD	1
ADAMS	1
TURNER	1
MARTIN	1
JAMES	1
MILLER	1
JONES	0
BLAKE	0
CLARK	0
FORD	0
SCOTT	0
KING	0

Поскольку вывод запроса IS_LEAF должен быть 0 или 1, к операции COUNT(*) необходимо применить функцию SIGN. В противном случае для концевых строк мы получим 14, а не 1. Альтернативно, для подсчета можно использовать таблицу только с одной строкой, т. к. нам нужно вернуть только 0 или 1. Например:

```
select e.ename,
       (select count(*) from t1 d
        where not exists
          (select null from emp f
           where f.mgr = e.empno)) as is_leaf
from emp e
order by 2 desc
```

ENAME	IS_LEAF
SMITH	1
ALLEN	1
WARD	1
ADAMS	1
TURNER	1
MARTIN	1

JAMES	1
MILLER	1
JONES	0
BLAKE	0
CLARK	0
FORD	0
SCOTT	0
KING	0

На следующем шаге определяем узлы ветвления (неконцевые узлы). Согласно определению, такими узлами являются строки служащих, которые имеют в своем подчинении других служащих и в то же время сами находятся в подчинении других служащих. Далее приводятся соответствующий скалярный подзапрос IS_BRANCH и его результаты:

```
select e.ename,
       (select sign(count(*)) from emp d
        where d.mgr = e.empno
          and e.mgr is not null) as is_branch
from emp e
order by 2 desc
```

ENAME	IS_BRANCH
JONES	1
BLAKE	1
SCOTT	1
CLARK	1
FORD	1
SMITH	0
TURNER	0
MILLER	0
JAMES	0
ADAMS	0
KING	0
ALLEN	0
MARTIN	0
WARD	0

Опять же, к операции COUNT(*) необходимо применить функцию SIGN. В противном случае для узлов ветвления можно потенциально получить значения, больше 1. Подобно скалярному подзапросу IS_LEAF, избежать необходимости использования функции SIGN можно с помощью таблицы с одной строкой. Далее приводится соответствующий запрос с использованием такой таблицы T1 и его результаты:

```
select e.ename,
       (select count(*) from t1 t
        where exists (
          select null from emp f
```



```

        where f.mgr = e.empno
              and e.mgr is not null)) as is_branch
from emp e
order by 2 desc

```

ENAME	IS_BRANCH
JONES	1
BLAKE	1
SCOTT	1
CLARK	1
FORD	1
SMITH	0
TURNER	0
MILLER	0
JAMES	0
ADAMS	0
KING	0
ALLEN	0
MARTIN	0
WARD	0

В завершение определяем корневые узлы. Согласно определению, таковыми будут строки служащих, которые являются руководителями, но не находятся ни в чьем подчинении. В таблице EMP корневым узлом является только строка служащего KING. Далее приводится соответствующий скалярный подзапрос IS_ROOT и результаты его исполнения:

```

select e.ename,
       (select sign(count(*)) from emp d
        where d.empno = e.empno
              and d.mgr is null) as is_root
from emp e
order by 2 desc

```

ENAME	IS_ROOT
KING	1
SMITH	0
ALLEN	0
WARD	0
JONES	0
TURNER	0
JAMES	0
MILLER	0
FORD	0
ADAMS	0
MARTIN	0
BLAKE	0
CLARK	0
SCOTT	0

Поскольку таблица EMP состоит всего лишь из 14 строк, то легко увидеть, что служащий KING является единственным корневым узлом, поэтому применять функцию SIGN к операции COUNT(*) нет особой надобности. Но в случае нескольких корневых узлов функцию SIGN, возможно, надо было бы применить или же, альтернативно, использовать подход со скалярным подзапросом к однострочной таблице, как в решениях для IS_BRANCH и IS_LEAF.

Oracle

Для версий Oracle, предшествующих Oracle Database 10g, применимо обсуждение решений, приведенное для других СУБД, т. к. эти решения будут работать без каких-либо модификаций и в Oracle. Для версий Oracle Database 10g и более поздних можно воспользоваться двумя функциями, которые значительно упрощают задачу определения корневых и концевых узлов. Это функции CONNECT_BY_ROOT и CONNECT_BY_ISLEAF, соответственно. На момент подготовки материала этой книги, чтобы использовать эти функции, выражение SQL должно содержать оператор CONNECT BY. На первом шаге определяем конечные узлы с помощью функции CONNECT_BY_ISLEAF:

```
select ename,
       connect_by_isleaf is_leaf
  from emp
 start with mgr is null
 connect by prior empno = mgr
 order by 2 desc
```

ENAME	IS_LEAF
-----	-----
ADAMS	1
SMITH	1
ALLEN	1
TURNER	1
MARTIN	1
WARD	1
JAMES	1
MILLER	1
KING	0
JONES	0
BLAKE	0
CLARK	0
FORD	0
SCOTT	0

Затем с помощью скалярного подзапроса определяем узлы ветвления. Вспомним, что узлы ветвления — это строки служащих, которые являются руководителями и одновременно находятся в чем-либо подчинении:

```

select ename,
       (select count(*) from emp e
        where e.mgr = emp.empno
              and emp.mgr is not null
              and rownum = 1) is_branch
from emp
start with mgr is null
connect by prior empno = mgr
order by 2 desc

```

ENAME	IS_BRANCH
-----	-----
JONES	1
SCOTT	1
BLAKE	1
FORD	1
CLARK	1
KING	0
MARTIN	0
MILLER	0
JAMES	0
TURNER	0
WARD	0
ADAMS	0
ALLEN	0
SMITH	0

Фильтр по ROWNUM необходим для того, чтобы обеспечить возвращение только значений 1 или 0 и никаких иных.

На последнем шаге определяем корневые узлы с помощью функции CONNECT_BY_ROOT. Решение находит значение ENAME для корневого узла и сравнивает его со всеми возвращенными запросом строками. При обнаружении совпадения с какой-либо строкой эта строка и признается корневым узлом:

```

select ename,
       decode(ename,connect_by_root(ename),1,0) is_root
from emp
start with mgr is null
connect by prior empno = mgr
order by 2 desc

```

ENAME	IS_ROOT
-----	-----
KING	1
JONES	0
SCOTT	0
ADAMS	0
FORD	0

```
SMITH          0
BLAKE          0
ALLEN          0
WARD           0
MARTIN         0
TURNER         0
JAMES          0
CLARK          0
MILLER         0
```

Функция SYS_CONNECT_BY_PATH формирует иерархию, начиная с корневого значения, как показано далее:

```
select ename,
       ltrim(sys_connect_by_path(ename,',')',') path
  from emp
start with mgr is null
connect by prior empno=mgr
```

ENAME	PATH
KING	KING
JONES	KING, JONES
SCOTT	KING, JONES, SCOTT
ADAMS	KING, JONES, SCOTT, ADAMS
FORD	KING, JONES, FORD
SMITH	KING, JONES, FORD, SMITH
BLAKE	KING, BLAKE
ALLEN	KING, BLAKE, ALLEN
WARD	KING, BLAKE, WARD
MARTIN	KING, BLAKE, MARTIN
TURNER	KING, BLAKE, TURNER
JAMES	KING, BLAKE, JAMES
CLARK	KING, CLARK
MILLER	KING, CLARK, MILLER

Чтобы найти строку корневого узла, просто извлекаем подстроку первого ENAME в PATH:

```
select ename,
       substr(root,1,instr(root,',')-1) root
  from (
select ename,
       ltrim(sys_connect_by_path(ename,',')',') root
  from emp
start with mgr is null
connect by prior empno=mgr
)
```

ENAME	ROOT
-----	-----
KING	
JONES	KING
SCOTT	KING
ADAMS	KING
FORD	KING
SMITH	KING
BLAKE	KING
ALLEN	KING
WARD	KING
MARTIN	KING
TURNER	KING
JAMES	KING
CLARK	KING
MILLER	KING

В завершение помечаем флагом значения столбца `ROOT` результата. Строка, для которой это значение равно `NULL`, является строкой корневого узла.

13.6. Подведем итоги

Широкое применение обобщенных табличных выражений во всех СУБД значительно упрощает стандартизацию иерархических запросов. Это громадный шаг вперед, поскольку иерархические отношения присутствуют во многих типах данных, даже в тех данных, в которых эти отношения не обязательно являются преднамеренными. Поэтому это обстоятельство должно учитываться при разработке запросов.

Нестандартные подходы

В этой главе рассматриваются типы запросов, которые не вошли в другие главы, потому что соответствующая им глава уже и так была достаточно обширна или потому что решаемые в них задачи относятся к разряду более развлекательных, чем практических. Это глава «чтобы поразвлечься». Вы можете никогда ни воспользоваться приведенными в ней рецептами, но тем не менее мы считаем эти рецепты интересными и решили включить их в книгу.

14.1. Создание кросс-табличных отчетов с помощью оператора SQL Server *PIVOT*

ЗАДАЧА

Требуется создать кросс-табличный отчет, в котором строки результирующего множества преобразуются в столбцы. Вы знакомы с традиционными методами транспонирования, но желали бы попробовать что-нибудь иное. В частности, вы хотите вернуть следующий набор результатов без использования выражений *CASE* или объединений:

DEPT_10	DEPT_20	DEPT_30	DEPT_40
3	5	6	0

РЕШЕНИЕ

Возвратить требуемое результирующее множество, не прибегая к использованию выражений *CASE* или дополнительных объединений, можно с помощью оператора *PIVOT*:

```

1 select [10] as dept_10,
2        [20] as dept_20,
3        [30] as dept_30,
4        [40] as dept_40
5   from (select deptno, empno from emp) driver
6 pivot (
7   count(driver.empno)
8   for driver.deptno in ( [10],[20],[30],[40] )
9 ) as empPivot
```

Обсуждение

На первый взгляд оператор `PIVOT` может выглядеть странно, но в решении он выполняет технически такую же операцию, что и более знакомый запрос транспонирования, показанный здесь:

```
select sum(case deptno when 10 then 1 else 0 end) as dept_10,
       sum(case deptno when 20 then 1 else 0 end) as dept_20,
       sum(case deptno when 30 then 1 else 0 end) as dept_30,
       sum(case deptno when 40 then 1 else 0 end) as dept_40
from emp
```

DEPT_10	DEPT_20	DEPT_30	DEPT_40
3	5	6	0

Чтобы разобраться, что, по существу, происходит, разложим действия оператора `PIVOT` на составляющие. Так, строка 5 решения содержит вложенный запрос `DRIVER`:

```
from (select deptno, empno from emp) driver
```

Псевдоним `DRIVER` использован по той причине, что строки представления (или табличного выражения), создаваемого этим вложенным запросом, подаются напрямую в операцию `PIVOT`. То есть имеется в виду, что указанные строки управляют (drive) этой операцией. Оператор `PIVOT` транспонирует строки в столбцы, обрабатывая элементы списка `FOR` в строке 8 решения:

```
for driver.deptno in ( [10],[20],[30],[40] )
```

Процесс обработки примерно следующий:

1. Для строк, в которых значение `DEPTNO` равно 10, выполняем операцию агрегации `COUNT(DRIVER.EMPNO)`.
2. Повторяем ее для строк со значением `DEPTNO`, равным 20, 30 и 40.

Элементы, приведенные в квадратных скобках в строке 8 решения, не только определяют значения, для которых выполняется операция агрегации, но и становятся названиями столбцов в результирующем множестве (уже без квадратных скобок). В операторе `SELECT` решения указываются ссылки на элементы списка `FOR` и может осуществляться присвоение им псевдонимов. Поскольку элементам списка `FOR` вы псевдонимы не присваиваете, эти элементы (без квадратных скобок) используются в качестве названий столбцов.

Все это достаточно интересно, поскольку такой простой вложенный запрос `DRIVER` можно сделать более сложным. Рассмотрим, например, ситуацию, когда нужно модифицировать результирующее множество таким образом, чтобы названиями столбцов стали настоящие названия отделов. Взглянем на содержимое таблицы `DEPT`:

```
select * from dept
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Мы хотим с помощью оператора `PIVOT` вернуть следующее результирующее множество:

ACCOUNTING	RESEARCH	SALES	OPERATIONS
3	5	6	0

Поскольку вложенный запрос `DRIVER` может быть практически любым табличным выражением, в нем можно разместить SQL-код для объединения таблицы `EMP` с таблицей `DEPT` с последующей обработкой полученных строк оператором `PIVOT`. Желаемое результирующее множество поможет вернуть следующий запрос:

```
select [ACCOUNTING] as ACCOUNTING,
       [SALES] as SALES,
       [RESEARCH] as RESEARCH,
       [OPERATIONS] as OPERATIONS
from (
    select d.dname, e.empno
    from emp e,dept d
    where e.deptno=d.deptno
) driver
pivot (
    count(driver.empno)
    for driver.dname in ([ACCOUNTING],[SALES],[RESEARCH],[OPERATIONS])
) as empPivot
```

Как можно видеть, оператор `PIVOT` предоставляет интересный метод для транспонирования результирующих множеств. Независимо от того, предпочтете ли вы этот метод более традиционным методам транспонирования, наличие дополнительного средства в инструментарии не мешает.

14.2. Обратное транспонирование кросс-табличных отчетов с помощью оператора SQL Server *UNPIVOT*

ЗАДАЧА

Имеется результирующее множество, полученное в результате транспонирования (или просто таблица каких-либо значений), для которого нужно отменить операцию транспонирования. Например, вместо результирующего множества из одной строки и четырех столбцов нужно вернуть таблицу с четырьмя строками и двумя столбцами. В частности, результирующее множество из предыдущего рецепта:

ACCOUNTING	RESEARCH	SALES	OPERATIONS
-----	-----	-----	-----
3	5	6	0

надо преобразовать в следующее результирующее множество:

DNAME	CNT
-----	-----
ACCOUNTING	3
RESEARCH	5
SALES	6
OPERATIONS	0

РЕШЕНИЕ

Вы же не думали, что SQL Server мог предоставить нам возможность транспонирования таблицы, не предоставив при этом способа транспонирования ее в обратном направлении? Чтобы выполнить обратное транспонирование таблицы, просто используем ее в качестве источника строк для оператора `UNPIVOT`, который выполнит всю необходимую работу. Нам нужно только указать требуемые названия столбцов:

```

1 select DNAME, CNT
2   from (
3     select [ACCOUNTING] as ACCOUNTING,
4           [SALES] as SALES,
5           [RESEARCH] as RESEARCH,
6           [OPERATIONS] as OPERATIONS
7     from (
8         select d.dname, e.empno
9         from emp e,dept d
10        where e.deptno=d.deptno
11
12        ) driver
13     pivot (
14         count(driver.empno)
15         for driver.dname in ([ACCOUNTING],[SALES],[RESEARCH],[OPERATIONS])
16     ) as empPivot
17 ) new_driver
18 unpivot (cnt for dname in (ACCOUNTING,SALES,RESEARCH,OPERATIONS)
19 ) as un_pivot

```

Мы полагаем, что, прежде чем разбираться с этим рецептом, вы ознакомились с предыдущим, поскольку код вложенного запроса `NEW_DRIVER` просто взят оттуда без каких бы то ни было изменений. (Если вы не понимаете, как он работает, разберитесь с предыдущим рецептом, прежде чем приступить к этому.)

Строки 3–16 кода нам уже знакомы, и новый синтаксис присутствует только в строке 18, где используется оператор `UNPIVOT`.

Этот оператор просто исследует результирующее множество, возвращаемое из `NEW_DRIVER`, и обрабатывает все его столбцы и строки. Например, когда при обработке названий столбцов возвращенного из `NEW_DRIVER` результирующего множества оператор `UNPIVOT` встречает название `ACCOUNTING`, он преобразовывает это название в значение строки в столбце `DNAME`. Кроме этого, он также возвращает значение `ACCOUNTING`, полученное из `NEW_DRIVER` (равное 3), как часть строки `ACCOUNTING` (в столбце `CNT`). Эти операции выполняются для каждого элемента, указанного в списке `FOR`, просто возвращая его в виде строки.

Полученный при выполнении этого запроса результат более компактный и состоит из двух столбцов (`DNAME` и `CNT`) и четырех строк:

```
select DNAME, CNT
  from (
    select [ACCOUNTING] as ACCOUNTING,
           [SALES] as SALES,
           [RESEARCH] as RESEARCH,
           [OPERATIONS] as OPERATIONS
    from (
      select d.dname, e.empno
      from emp e,dept d
      where e.deptno=d.deptno
    ) driver
    pivot (
      count(driver.empno)
      for driver.dname in ( [ACCOUNTING],[SALES],[RESEARCH],[OPERATIONS] )
    ) as empPivot
  ) new_driver
unpivot (cnt for dname in (ACCOUNTING,SALES,RESEARCH,OPERATIONS)
) as un_pivot
```

DNAME	CNT
ACCOUNTING	3
RESEARCH	5
SALES	6
OPERATIONS	0

14.3. Транспонирование результирующего множества с помощью оператора Oracle *MODEL*

ЗАДАЧА

Так же, как и в первом рецепте этой главы, мы хотим найти альтернативу традиционным способам выполнения транспонирования, которые нам известны к этому моменту. В частности, мы хотим испытать оператор Oracle `MODEL`. Но, в отличие от оператора SQL Server `PIVOT`, оператор Oracle `MODEL` не предназначен для транспонирования строк результирующего множества. Более того, честно говоря, транспони-

рование с применением оператора `MODEL` является примером неправильного его использования и не по задуманному предназначению. Тем не менее оператор `MODEL` делает возможным интересный подход к решению такой распространенной задачи. В нашем случае мы хотим преобразовать показанное далее результирующее множество:

```
select deptno, count(*) cnt
       from emp
       group by deptno
```

DEPTNO	CNT
10	3
20	5
30	6

в следующее:

D10	D20	D30
3	5	6

РЕШЕНИЕ

В операторе `MODEL` используем агрегацию и выражения `CASE` таким же образом, как и при транспонировании с помощью обычных методов. Но в нашем случае основное различие состоит в использовании массивов для хранения результатов агрегации с их возвращением в результирующем множестве:

```
select max(d10) d10,
       max(d20) d20,
       max(d30) d30
       from (
select d10,d20,d30
       from ( select deptno, count(*) cnt from emp group by deptno )
       model
       dimension by(deptno d)
       measures(deptno, cnt d10, cnt d20, cnt d30)
       rules(
         d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
         d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
         d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
       )
       )
```

Обсуждение

Оператор `MODEL` — мощное новое средство в инструментарии SQL Oracle. Начав работать с ним, вы обнаружите такие полезные возможности, как итерация, доступ к значениям строк с помощью массивов, процедура «`upsert`» — обновление

(UPDATE) строк, если они существуют, или вставка их, если нет (INSERT), а также создание справочных моделей. Как вы сможете увидеть, в этом рецепте никакие из указанных замечательных возможностей оператора MODEL не используются. Но разве это не интересно — суметь подойти к решению задачи с разных сторон и использовать имеющиеся возможности нестандартными способами (хотя бы для того, чтобы понять, когда определенные возможности более полезны, чем другие)?

Первое, что нужно сделать, чтобы понять предложенное решение, — это проанализировать вложенный запрос в конструкции FROM. Запрос просто подсчитывает количество служащих в каждом отделе DEPTNO таблицы EMP:

```
select deptno, count(*) cnt
      from emp
      group by deptno
```

DEPTNO	CNT
10	3
20	5
30	6

Это результирующее множество передается для обработки оператору MODEL. В конструкции MODEL можно увидеть три основных составляющих его блока: DIMENSION BY, MEASURES и RULES. Начнем рассмотрение их работы с блока MEASURES.

Элементы в списке MEASURES — просто четыре массива, объявленные для этого запроса: DEPTNO, D10, D20 и D30. Подобно названиям столбцов в списке SELECT, массивам в списке MEASURES тоже можно присваивать псевдонимы. Как можно видеть, три из этих четырех массивов являются значениями поля CNT вложенного запроса.

Если список MEASURES содержит массивы, то элементы в операторе DIMENSION BY являются индексами этих массивов. Учтите следующее: массив D10 — это просто псевдоним для CNT. Посмотрев на результирующее множество приведенного в *разд. «Задача»* запроса, вы увидите, что CNT имеет три значения: 3, 5 и 6. Таким образом, создавая массив значений CNT, мы создаем массив из трех элементов — в частности, из трех целых чисел: 3, 5 и 6. Но как обращаться к индивидуальным значениям этого массива? С помощью индекса массива. Определенный в подоператоре DIMENSION BY индекс имеет три значения: 10, 20 и 30 (из приведенного ранее результирующего множества). Таким образом, значение следующего выражения:

```
d10[10]
```

будет 3, т. к. мы обращаемся к значению CNT в массиве D10 для DEPTNO 10 (которое равно 3).

Поскольку все три массива (D10, D20, D30) содержат значения CNT, все они имеют одинаковые результаты. Как же тогда нам правильно подсчитать правильный массив? С помощью подоператора RULES. Посмотрев на результирующее множество приведенного ранее вложенного запроса, можно увидеть, что DEPTNO имеет три значения: 10, 20 и 30. Выражения с применением CASE в подоператоре RULES просто определяют каждое значение в массиве DEPTNO:

- ◆ если значение равно 10, сохраняем CNT для DEPTNO 10 в D10[10], в противном случае сохраняем 0;
- ◆ если значение равно 20, сохраняем CNT для DEPTNO 20 в D20[20], в противном случае сохраняем 0;
- ◆ если значение равно 30, сохраняем CNT для DEPTNO 30 в D30[30], в противном случае сохраняем 0.

Если все это вызывает у вас чувство, испытываемое Алисой, падающей в заячью нору, не переживайте. Отложите попытку подробно разобраться на потом, а сейчас просто возьмите и выполните все рассмотренное на этом этапе. Приведенное далее результирующее множество представляет обсуждаемые моменты. Иногда вопрос легче понять, прочитав немного его объяснение, затем просмотреть код, осуществляющий прочитанное, после чего снова возвратиться к чтению объяснения. Следующий далее код достаточно простой, стоит только увидеть его в действии:

```
select deptno, d10,d20,d30
  from ( select deptno, count(*) cnt from emp group by deptno )
model
  dimension by(deptno d)
  measures(deptno, cnt d10, cnt d20, cnt d30)
  rules(
    d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
    d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
    d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
  )
```

DEPTNO	D10	D20	D30
10	3	0	0
20	0	5	0
30	0	0	6

Как можно видеть, значения в каждом массиве изменяются именно с помощью подоператора RULES. Если и до сих пор не все понятно, просто исполните этот запрос опять, но прокомментируйте выражение в подоператоре RULES, как выделено далее курсивом:

```
select deptno, d10,d20,d30
  from ( select deptno, count(*) cnt from emp group by deptno )
model
  dimension by(deptno d)
  measures(deptno, cnt d10, cnt d20, cnt d30)
  rules(
    /*
    d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
    d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
    d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
    */
  )
```

DEPTNO	D10	D20	D30
10	3	3	3
20	5	5	5
30	6	6	6

Теперь должно быть ясно, что результирующее множество оператора `MODEL` такое же, как и создаваемое вложенным запросом, с единственной разницей, что результатам операции `COUNT` присваиваются псевдонимы `D10`, `D20` и `D30`. Следующий запрос доказывает это:

```
select deptno, count(*) d10, count(*) d20, count(*) d30
  from emp
 group by deptno
```

DEPTNO	D10	D20	D30
10	3	3	3
20	5	5	5
30	6	6	6

Таким образом, оператор `MODEL` всего лишь берет значения для `DEPTNO` и `CNT` и помещает их в массивы, а затем обеспечивает представление каждым массивом только одного `DEPTNO`. На этом этапе каждый из массивов `D10`, `D20` и `D30` содержит только одно ненулевое значение, представляющее значение `CNT` для соответствующего `DEPTNO`. Результирующее множество уже транспонировано, и осталось только применить агрегатную функцию `MAX` (также можно было бы использовать функцию `MIN` или `SUM` — в нашем случае не было бы никакой разницы), чтобы вернуть только одну строку:

```
select max(d10) d10,
       max(d20) d20,
       max(d30) d30
  from (
select d10,d20,d30
  from ( select deptno, count(*) cnt from emp group by deptno )
 model
  dimension by (deptno d)
  measures (deptno, cnt d10, cnt d20, cnt d30)
  rules (
    d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
    d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
    d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
  )
)
```

D10	D20	D30
3	5	6

14.4. Извлечение элементов строки, положение которых неизвестно

ЗАДАЧА

Имеется строковое поле, содержащее сериализованные данные. Надо выполнить парсинг строки и извлечь из нее требуемую информацию. К сожалению, эта информация может находиться в разных местах строки, расположение которых неизвестно. Но, к счастью, требуемая информация обрамлена определенными символами, по которым можно определить ее местонахождение. Рассмотрим, например, следующие строки:

```
xxxxxabc[867]xxx[-]xxxx[5309]xxxxx
xxxxxtime:[11271978]favnum:[4]id:[Joe]xxxxx
call:[F_GET_ROWS()]b1:[ROSEWOOD...SIR]b2:[44400002]77.90xxxxx
film:[non_marked]qq:[unit]tailpipe:[withabanana?]80sxxxxx
```

Нам нужно извлечь значения в квадратных скобках и получить следующее результирующее множество:

FIRST_VAL	SECOND_VAL	LAST_VAL
867	-	5309
11271978	4	Joe
F_GET_ROWS()	ROSEWOOD...SIR	44400002
non_marked	unit	withabanana?

РЕШЕНИЕ

Несмотря на то что мы не знаем точного местонахождения требуемых нам значений, мы знаем, что они обрамлены квадратными скобками [], а также знаем, что каждая исходная строка содержит три таких значения. Расположение квадратных скобок в строке находим с помощью встроенной функции Oracle `INSTR`, а обрамленные ими значения извлекаем с помощью встроенной функции `SUBSTR`. Строки для парсинга будут содержаться в представлении `V` (используется просто для удобства чтения), создаваемом следующим запросом:

```
create view V
as
select 'xxxxxabc[867]xxx[-]xxxx[5309]xxxxx' msg
  from dual
 union all
select 'xxxxxtime:[11271978]favnum:[4]id:[Joe]xxxxx' msg
  from dual
 union all
select 'call:[F_GET_ROWS()]b1:[ROSEWOOD...SIR]b2:[44400002]77.90xxxxx' msg
  from dual
 union all
```

```
select 'film:[non_marked]qq:[unit]tailpipe:[withabanana?]80sxxxxx' msg
from dual
```

```
1 select substr(msg,
2     instr(msg, '[' ,1,1)+1,
3     instr(msg, ']' ,1,1)-instr(msg, '[' ,1,1)-1) first_val,
4     substr(msg,
5     instr(msg, '[' ,1,2)+1,
6     instr(msg, ']' ,1,2)-instr(msg, '[' ,1,2)-1) second_val,
7     substr(msg,
8     instr(msg, '[' ,-1,1)+1,
9     instr(msg, ']' ,-1,1)-instr(msg, '[' ,-1,1)-1) last_val
10 from V
```

Обсуждение

Благодаря встроенной функции Oracle `INSTR` решение этой задачи значительно упрощается. Так как мы знаем, что значения заключены в квадратные скобки и что таких скобок имеется три пары, на первом шаге решения надо просто использовать функцию `INSTR`, чтобы определить числовые позиции пар квадратных скобок в каждой строке. Следующий запрос возвращает числовые позиции открывающих и закрывающих квадратных скобок в каждой строке:

```
select instr(msg, '[' ,1,1) "1st_[" ,
       instr(msg, ']' ,1,1) "]"_1st",
       instr(msg, '[' ,1,2) "2nd_[" ,
       instr(msg, ']' ,1,2) "]"_2nd",
       instr(msg, '[' ,-1,1) "3rd_[" ,
       instr(msg, ']' ,-1,1) "]"_3rd"
from V
```

1st_ []_1st	2nd_ []_2nd	3rd_ []_3rd
9 13	17 19	24 29
11 20	28 30	34 38
6 19	23 38	42 51
6 17	21 26	36 49

Что ж, вся трудная работа выполнена. Осталось только вставить эти числовые позиции в функцию `SUBSTR`, чтобы извлечь из `MSG` подстроки, находящиеся в этих позициях. Можно видеть, что в полном решении значения, возвращаемые функцией `INSTR`, подвергаются некоторым простым арифметическим операциям — в частности, к ним добавляется и вычитается 1, чтобы не допустить возвращения в результирующем множестве открывающей квадратной скобки. Далее приводится решение без добавления и вычитания 1 из возвращаемых функцией `INSTR` значений. Обратите внимание на открывающую скобку в каждом значении результирующего множества:


```

select substr(msg,
             instr(msg, '[' ,1,1) ,
             instr(msg, ']' ,1,1)-instr(msg, '[' ,1,1)) first_val,
       substr(msg,
             instr(msg, '[' ,1,2) ,
             instr(msg, ']' ,1,2)-instr(msg, '[' ,1,2)) second_val,
       substr(msg,
             instr(msg, '[' ,-1,1) ,
             instr(msg, ']' ,-1,1)-instr(msg, '[' ,-1,1)) last_val
from V

```

FIRST_VAL	SECOND_VAL	LAST_VAL
[867	[-	[5309
[11271978	[4	[Joe
[F_GET_ROWS()	[ROSEWOOD...SIR	[44400002
[non_marked	[unit	[withabanana?

Как можно видеть, в приведенном здесь результирующем множестве вместе со значениями возвращаются и открывающие квадратные скобки. Возможно, вы думаете: «Хорошо. Вернем добавление 1 к INSTR и избавимся от ведущей квадратной скобки. Но зачем нам вычитать 1?» Потому что, если вернуть добавление, но убрать вычитание, в возвращаемые значения будут включаться закрывающие квадратные скобки:

```

select substr(msg,
             instr(msg, '[' ,1,1)+1,
             instr(msg, ']' ,1,1)-instr(msg, '[' ,1,1)) first_val,
       substr(msg,
             instr(msg, '[' ,1,2)+1,
             instr(msg, ']' ,1,2)-instr(msg, '[' ,1,2)) second_val,
       substr(msg,
             instr(msg, '[' ,-1,1)+1,
             instr(msg, ']' ,-1,1)-instr(msg, '[' ,-1,1)) last_val
from V

```

FIRST_VAL	SECOND_VAL	LAST_VAL
867]	-]	5309]
11271978]	4]	Joe]
F_GET_ROWS()]	ROSEWOOD...SIR]	44400002]
non_marked]	unit]	withabanana?]

К этому моменту должно быть ясно: чтобы обеспечить отсутствие обеих квадратных скобок, к начальному индексу нужно добавить 1, а из конечного — вычесть 1.

14.5. Вычисление количества дней в году (альтернативное решение для Oracle)

ЗАДАЧА

Требуется определить количество дней в году.



В этом рецепте представлено решение, альтернативное решению задачи из рецепта 9.2, годящееся только для Oracle.

РЕШЕНИЕ

Используем функцию `TO_CHAR`, чтобы представить дату последнего дня года в виде трехзначного значения порядкового номера дня года:

```
1 select 'Days in 2021: '||
2         to_char(add_months(trunc(sysdate, 'y'), 12)-1, 'DDD')
3         as report
4     from dual
5  union all
6 select 'Days in 2020: '||
7         to_char(add_months(trunc(
8         to_date('01-SEP-2020'), 'y'), 12)-1, 'DDD')
9         from dual
```

REPORT

```
-----
Days in 2021: 365
Days in 2020: 366
```

Обсуждение

На первом шаге определяем начальную дату года, используя для этого функцию `TRUNC`:

```
select trunc(to_date('01-SEP-2020'), 'y')
       from dual
```

```
TRUNC(TO_DA
-----
01-JAN-2020
```

Затем с помощью функции `ADD_MONTH` добавляем к усеченной таким образом дате один год (12 месяцев). Далее вычитаем один день, получая конечную дату года исходной даты:

```
select add_months(
         trunc(to_date('01-SEP-2020'), 'y'),
         12) before_subtraction,
```

```

add_months(
    trunc(to_date('01-SEP-2020'),'Y'),
    12)-1 after_subtraction
from dual

```

```

BEFORE_SUBT AFTER_SUBTR
-----
01-JAN-2021 31-DEC-2020

```

Получив дату последнего дня рассматриваемого года, просто применяем к ней функцию `TO_CHAR`, чтобы вернуть трехзначный порядковый номер (1-й, 50-й и т. д.) последнего дня этого года:

```

select to_char(
    add_months(
        trunc(to_date('01-SEP-2020'),'Y'),
        12)-1,'DDD') num_days_in_2020
from dual

```

```

NUM
---
366

```

14.6. Поиск смешанных буквенно-цифровых строк

ЗАДАЧА

Имеется столбец, содержащий смешанные буквенно-цифровые значения. Требуется вернуть только те строки, которые содержат как буквенные, так и цифровые символы. Иными словами, строки, содержащие или только цифры, или только буквы, не возвращаются. Например, для следующих данных:

```

STRINGS
-----
1010 switch
333
3453430278
ClassSummary
findRow 55
threes

```

нужно получить такое конечное результирующее множество:

```

STRINGS
-----
1010 switch
findRow 55

```

РЕШЕНИЕ

С помощью встроенной функции `TRANSLATE` преобразовываем все буквы и цифры исходной строки в соответствующие специальные символы. Затем возвращаем

только те строки, которые содержат по крайней мере по одному вхождению каждого из этих двух символов. В предложенном решении используется синтаксис Oracle, но как DB2, так и PostgreSQL поддерживают функцию `TRANSLATE`, и откорректировать решения под эти платформы не представит никаких сложностей:

```
with v as (
select 'ClassSummary' strings from dual union
select '3453430278' from dual union
select 'findRow 55' from dual union
select '1010 switch' from dual union
select '333' from dual union
select 'threes' from dual
)
select strings
      from (
select strings,
translate(
strings,
'abcdefghijklmnopqrstuvwxyz0123456789',
rpad('#',26,'#')||rpad('*',10,'*')) translated
      from v
) x
where instr(translated,'#') > 0
and instr(translated,'*') > 0
```



Как альтернативу оператору `WITH` можно использовать вложенный запрос для создания представления или просто создать представление отдельно.

Обсуждение

Наличие функции `TRANSLATE` делает решение этой задачи исключительно легким. На первом шаге с помощью этой функции заменяем все буквы символами `#`, а цифры — символами `*`. Далее приведены результаты этой операции, выполняемой вложенным запросом X:

```
with v as (
select 'ClassSummary' strings from dual union
select '3453430278' from dual union
select 'findRow 55' from dual union
select '1010 switch' from dual union
select '333' from dual union
select 'threes' from dual
)
select strings,
      translate(
strings,
```

```

'abcdefghijklmnopqrstuvwxyz0123456789',
rpad('#',26,'#')||rpad('*',10,'') translated
from v

```

STRINGS	TRANSLATED
1010 switch	**** #####
333	***
3453430278	*****
ClassSummary	C####S#####
findRow 55	####R## **
threes	#####

Затем надо просто выбрать только те строки, которые содержат по крайней мере по одному вхождению каждого из этих двух символов. Наличие символов # и * в строке определяем с помощью функции INSTR. Если строка содержит эти два символа, возвращаемое функцией значение будет больше нуля. Далее показаны строки конечного результата (вместе с их соответствующими преобразованными версиями для ясности):

```

with v as (
select 'ClassSummary' strings from dual union
select '3453430278' from dual union
select 'findRow 55' from dual union
select '1010 switch' from dual union
select '333' from dual union
select 'threes' from dual
)
select strings, translated
from (
select strings,
translate(
strings,
'abcdefghijklmnopqrstuvwxyz0123456789',
rpad('#',26,'#')||rpad('*',10,'')) translated
from v
)
where instr(translated,'#') > 0
and instr(translated,'*') > 0

```

STRINGS	TRANSLATED
1010 switch	**** #####
findRow 55	####R## **

14.7. Преобразование десятичных чисел в двоичные в Oracle

ЗАДАЧА

Требуется преобразовать целые десятичные числа в соответствующие двоичные в СУБД Oracle. Например, надо вернуть все значения зарплат в таблице EMP в двоичном представлении, как показано в следующем результирующем множестве:

ENAME	SAL	SAL_BINARY
SMITH	800	1100100000
ALLEN	1600	11001000000
WARD	1250	10011100010
JONES	2975	101110011111
MARTIN	1250	10011100010
BLAKE	2850	101100100010
CLARK	2450	100110010010
SCOTT	3000	101110111000
KING	5000	1001110001000
TURNER	1500	10111011100
ADAMS	1100	10001001100
JAMES	950	1110110110
FORD	3000	101110111000
MILLER	1300	10100010100

РЕШЕНИЕ

Благодаря своей возможности выполнять итерации и обеспечивать доступ к значениям строк через элементы массива для решения этой задачи идеально подходит функция `MODEL`. (Предполагаем, что мы вынуждены применять SQL, поскольку в рассматриваемом случае пользовательская функция была бы более уместна.) Подобно другим решениям этой книги, даже если вы не предполагаете, что сможете найти практическое применение этому решению, сфокусируйтесь на применяемом в нем подходе. Полезно знать, что оператор `MODEL` может выполнять процедурные задачи, сохраняя при этом свойства и мощь SQL, направленные на работы с множествами. Поэтому, даже если вы думаете: «Я бы никогда не делал так в SQL», ничего страшного. Мы никоим образом не говорим, что вы должны это делать или не должны. Мы просто предлагаем сосредоточиться на используемом подходе, чтобы суметь применить его для решения более «практичной», на ваш взгляд, задачи.

Следующее решение возвращает все значения `ENAME` и `SAL` из таблицы `EMP`, при этом вызывая в скалярном подзапросе оператор `MODEL` (таким образом, оно является работающей с таблицей `EMP` своеобразной свободной функцией, которая просто принимает входные данные, обрабатывает их и возвращает значение, во многом подобно тому, как это делала бы обычная функция):

```

1 select ename,
2     sal,
3     (
4     select bin
5     from dual
6     model
7     dimension by ( 0 attr )
8     measures ( sal num,
9                cast(null as varchar2(30)) bin,
10               '0123456789ABCDEF' hex
11            )
12     rules iterate (10000) until (num[0] <= 0) (
13       bin[0] = substr(hex[cv()],mod(num[cv()],2)+1,1)||bin[cv()],
14       num[0] = trunc(num[cv()]/2)
15     )
16   ) sal_binary
17 from emp

```

Обсуждение

Как упоминалось в *разд. «Решение»*, для решения этой задачи, вероятнее всего, лучше подошла бы пользовательская функция. Собственно говоря, идея для этого рецепта и возникла из функции. По сути, предлагаемый рецепт является адаптацией функции `TO_BASE`, разработанной Томом Кайтом (Tom Kyte) из корпорации Oracle. Подобно другим рецептам этой книги, которым вы можете не найти применения, даже если вы решите не использовать этот рецепт, он хорошо демонстрирует такие возможности оператора `MODEL`, как выполнение итераций и доступ к строкам через элементы массива.

Чтобы упростить объяснение, рассмотрим слегка измененный подзапрос, содержащий оператор `MODEL`. Следующий код практически такой же, как и в подзапросе решения, с тем исключением, что в нем жестко запрограммировано возвращение значения 2 в двоичном формате:

```

select bin
  from dual
 model
 dimension by ( 0 attr )
 measures ( 2 num,
           cast(null as varchar2(30)) bin,
           '0123456789ABCDEF' hex
        )
 rules iterate (10000) until (num[0] <= 0) (
   bin[0] = substr (hex[cv()],mod(num[cv()],2)+1,1)||bin[cv()],
   num[0] = trunc(num[cv()]/2)
 )

```

BIN

10

Следующий запрос выдает значения, возвращаемые после одной итерации блока RULES, определенного в предыдущем запросе:

```
select 2 start_val,
       '0123456789ABCDEF' hex,
       substr('0123456789ABCDEF',mod(2,2)+1,1) ||
       cast(null as varchar2(30)) bin,
       trunc(2/2) num
from dual
```

START_VAL	HEX	BIN	NUM
2	0123456789ABCDEF	0	1

Переменная `START_VAL` определяет число, которое нужно преобразовать в двоичное представление. В нашем случае это 2. Значение `BIN` получается в результате операции `SUBSTR` над строкой `0123456789ABCDEF` (в исходном решении `HEX`). Значение `NUM` задает условие для выхода из цикла.

Как можно видеть из предыдущего результирующего множества, после первой итерации цикла значение `BIN` равно 0, а `NUM` — 1. Так как значение `NUM` не меньше или не равно 0, выполняется другая итерация цикла. Результаты этой итерации представлены показанным далее выражением SQL:

```
select num start_val,
       substr('0123456789ABCDEF',mod(1,2)+1,1) || bin bin,
       trunc(1/2) num
from (
select 2 start_val,
       '0123456789ABCDEF' hex,
       substr('0123456789ABCDEF',mod(2,2)+1,1) ||
       cast(null as varchar2(30)) bin,
       trunc(2/2) num
from dual
)
```

START_VAL	BIN	NUM
1	10	0

После следующей итерации цикла результат обработки функции `SUBSTR` значения `HEX` будет 1, к которой присоединяется предыдущее значение `BIN` — 0. Проверочное значение `NUM` теперь равно 0. Таким образом, это была последняя итерация, и возвращенное значение 10 является двоичным представлением десятичного числа 2. Разобравшись с происходящим, из оператора `MODEL` можно убрать итерацию и выполнить его пошагово, чтобы проследить, каким образом применяются правила для получения конечного результирующего множества. Далее приводится соответствующий запрос и его результаты:

```
select 2 orig_val, num, bin
from dual
```



```

model
dimension by ( 0 attr )
measures ( 2 num,
           cast(null as varchar2(30)) bin,
           '0123456789ABCDEF' hex
           )
rules (
  bin[0] = substr (hex[cv()],mod(num[cv()],2)+1,1) || bin[cv()],
  num[0] = trunc(num[cv()]/2),
  bin[1] = substr (hex[0],mod(num[0],2)+1,1) || bin[0],
  num[1] = trunc(num[0]/2)
)

```

```

ORIG_VAL NUM BIN
-----
      2   1  0
      2   0 10

```

14.8. Транспонирование ранжированного результирующего множества

ЗАДАЧА

Требуется присвоить ранги значениям таблицы, а затем транспонировать полученное результирующее множество в три столбца. В частности, в первом столбце надо отобразить первые три наибольшие значения, во втором — следующие три, в третьем — все остальные. Например, нужно выполнить ранжирование служащих в таблице по зарплате SAL, а затем транспонировать результаты в три столбца. Результирующее множество должно выглядеть следующим образом:

TOP_3	NEXT_3	REST
KING (5000)	BLAKE (2850)	TURNER (1500)
FORD (3000)	CLARK (2450)	MILLER (1300)
SCOTT (3000)	ALLEN (1600)	MARTIN (1250)
JONES (2975)		WARD (1250)
		ADAMS (1100)
		JAMES (950)
		SMITH (800)

РЕШЕНИЕ

Ключ к решению этой задачи — использовать для ранжирования служащих по столбцу SAL оконную функцию DENSE_RANK OVER, допуская при этом наличие дубликатов рангов для одинаковых значений SAL. Таким способом можно легко выделить первые три наибольшие зарплаты, следующие три, а затем все остальные.

Затем с помощью оконной функции `ROW_NUMBER OVER` ранжируем служащих в группах (группа трех наибольших зарплат, группа следующих трех или группа всех остальных). После чего просто выполняем классическое транспонирование, используя в процессе встроенные строковые функции соответствующей СУБД для форматирования результатов. В представленном далее решении используется синтаксис Oracle. Однако, поскольку все рассматриваемые СУБД поддерживают оконные функции, его с легкостью можно откорректировать для работы на этих платформах:

```

1 select max(case grp when 1 then rpad(ename,6) ||
2           ' (|| sal ||)' end) top_3,
3         max(case grp when 2 then rpad(ename,6) ||
4           ' (|| sal ||)' end) next_3,
5         max(case grp when 3 then rpad(ename,6) ||
6           ' (|| sal ||)' end) rest
7   from (
8 select ename,
9        sal,
10       rnk,
11       case when rnk <= 3 then 1
12            when rnk <= 6 then 2
13            else 3
14       end grp,
15       row_number()over (
16         partition by case when rnk <= 3 then 1
17                       when rnk <= 6 then 2
18                       else 3
19         end
20         order by sal desc, ename
21       ) grp_rnk
22   from (
23 select ename,
24        sal,
25        dense_rank()over(order by sal desc) rnk
26   from emp
27        ) x
28        ) y
29  group by grp_rnk

```

Обсуждение

Этот рецепт — прекрасный пример того, как можно многого достичь с помощью оконных функций. На первый взгляд решение может выглядеть сложным, но если рассмотреть его работу подробно, вы будете удивлены, насколько оно простое. Начнем с исполнения вложенного запроса X:

```

select ename,
       sal,
       dense_rank()over(order by sal desc) rnk
from emp

```

ENAME	SAL	RNK
KING	5000	1
SCOTT	3000	2
FORD	3000	2
JONES	2975	3
BLAKE	2850	4
CLARK	2450	5
ALLEN	1600	6
TURNER	1500	7
MILLER	1300	8
WARD	1250	9
MARTIN	1250	9
ADAMS	1100	10
JAMES	950	11
SMITH	800	12

Как можно видеть из полученного результирующего множества, вложенный запрос X просто ранжирует служащих по значению SAL, при этом допуская дубликаты рангов. (Поскольку используется функция DENSE_RANK, а не просто RANK, получается непрерывная последовательность рангов.) На следующем шаге группируем строки, возвращенные вложенным запросом X, сравнивая ранги, присвоенные функцией DENSE_RANK, с помощью выражения CASE. Кроме этого, с помощью функции ROW_NUMBER OVER также ранжируем служащих по SAL в каждой создаваемой выражением CASE группе. Все это происходит во вложенном запросе Y, чей код и результаты приводятся далее:

```
select ename,
       sal,
       rnk,
       case when rnk <= 3 then 1
            when rnk <= 6 then 2
            else 3
       end grp,
       row_number()over (
         partition by case when rnk <= 3 then 1
                      when rnk <= 6 then 2
                      else 3
         end
         order by sal desc, ename
       ) grp_rnk
from (
select ename,
       sal,
       dense_rank()over(order by sal desc) rnk
from emp
) x
```

ENAME	SAL	RNK	GRP	GRP_RNK
KING	5000	1	1	1
FORD	3000	2	1	2
SCOTT	3000	2	1	3
JONES	2975	3	1	4
BLAKE	2850	4	2	1
CLARK	2450	5	2	2
ALLEN	1600	6	2	3
TURNER	1500	7	3	1
MILLER	1300	8	3	2
MARTIN	1250	9	3	3
WARD	1250	9	3	4
ADAMS	1100	10	3	5
JAMES	950	11	3	6
SMITH	800	12	3	7

Теперь начинает проясняться полный запрос, и если вы следовали его рассмотрению с самого начала (от вложенного запроса X), вы должны видеть, что он не такой и сложный. На этом этапе для каждого служащего запрос возвращает следующие данные:

- ◆ зарплату SAL;
- ◆ ранг RNK его зарплаты SAL по сравнению с другими служащими;
- ◆ номер группы GRP, в которую помещен этот служащий на основании ранга его зарплаты SAL;
- ◆ ранг в рамках группы GRP_RNK, опять же на основании значения SAL.

Теперь выполняем обычное транспонирование по столбцу ENAME, используя оператор конкатенации Oracle || для присоединения значения зарплаты SAL. Функция RPAD обеспечивает аккуратное выравнивание числовых значений в скобках. Наконец, применяем функцию GROUP BY по значению GRP_RNK, чтобы обеспечить наличие в результирующем множестве всех служащих. Полученное конечное результирующее множество показано далее:

```
select max(case grp when 1 then rpad(ename,6) ||
          ' (|| sal ||)' end) top_3,
max(case grp when 2 then rpad(ename,6) ||
          ' (|| sal ||)' end) next_3,
max(case grp when 3 then rpad(ename,6) ||
          ' (|| sal ||)' end) rest
  from (
select ename,
       sal,
       rnk,
       case when rnk <= 3 then 1
            when rnk <= 6 then 2
            else 3
       end grp,
```

```

row_number() over (
  partition by case when rnk <= 3 then 1
                when rnk <= 6 then 2
                else 3
  end
  order by sal desc, ename
) grp_rnk
from (
select ename,
       sal,
       dense_rank() over (order by sal desc) rnk
from emp
  ) x
  ) y
group by grp_rnk

```

TOP_3	NEXT_3	REST
KING (5000)	BLAKE (2850)	TURNER (1500)
FORD (3000)	CLARK (2450)	MILLER (1300)
SCOTT (3000)	ALLEN (1600)	MARTIN (1250)
JONES (2975)		WARD (1250)
		ADAMS (1100)
		JAMES (950)
		SMITH (800)

Проанализировав запросы на всех этапах, можно заметить, что обращение к таблице осуществляется только один раз. Оконные функции замечательны, среди прочего, тем большим объемом работы, который они могут выполнить лишь за одно обращение к данным. Благодаря им нет необходимости выполнять самообъединения или использовать временные таблицы — просто обращаемся к требуемым строкам и предоставляем оконным функциям делать все остальное. В нашем случае обращаться к таблице EMP нужно только во вложенном запросе X, после чего остается лишь отформатировать результирующее множество, чтобы придать ему требуемый вид. Вдумайтесь, что это означает в терминах производительности, если для создания отчета такого типа нужно лишь одно обращение к таблице. Довольно эффективно, не так ли?

14.9. Добавление заголовка столбца в дважды развернутое результирующее множество

ЗАДАЧА

Требуется сложить вместе два результирующих множества, а затем транспонировать полученный результат в два столбца. Кроме этого, каждую группу строк в каждом столбце надо снабдить «заголовком». Например, у нас есть две таблицы,

содержащие информацию о служащих, работающих в двух разных подразделениях компании (скажем, в исследовательском и производственном):

```
select * from it_research
```

```
DEPTNO ENAME
-----
100 HOPKINS
100 JONES
100 TONEY
200 MORALES
200 P.WHITAKER
200 MARCIANO
200 ROBINSON
300 LACY
300 WRIGHT
300 J.TAYLOR
```

```
select * from it_apps
```

```
DEPTNO ENAME
-----
400 CORRALES
400 MAYWEATHER
400 CASTILLO
400 MARQUEZ
400 MOSLEY
500 GATTI
500 CALZAGHE
600 LAMOTTA
600 HAGLER
600 HEARNS
600 FRAZIER
700 GUINN
700 JUDAH
700 MARGARITO
```

Нам надо создать отчет, расположив в нем служащих из обеих таблиц в двух столбцах, соответствующих подразделениям. В частности, в каждом столбце нужно вернуть номер отдела DEPTNO и имена ENAME работающих в нем служащих. Конечное результирующее множество должно иметь следующий вид:

RESEARCH	APPS
-----	-----
100	400
JONES	MAYWEATHER
TONEY	CASTILLO
HOPKINS	MARQUEZ
200	MOSLEY
P.WHITAKER	CORRALES

MARCIANO	500
ROBINSON	CALZAGHE
MORALES	GATTI
300	600
WRIGHT	HAGLER
J.TAYLOR	HEARNS
LACY	FRAZIER
	LAMOTTA
	700
	JUDAH
	MARGARITO
	GUINN

РЕШЕНИЕ

По существу, для решения этой задачи надо лишь выполнить простое объединение с последующим транспонированием, но с дополнительной модификацией: для всех возвращаемых ENAME каждого отдела нужно указывать соответствующее значение DEPTNO в начале группы. В решении используется декартово произведение для создания дополнительной строки для каждого DEPTNO, чтобы у нас было достаточно строк для отображения всех служащих отдела плюс строка для вывода соответствующего DEPTNO. Решение основано на синтаксисе Oracle, но поскольку DB2 поддерживает оконные функции для вычисления скользящих окон (оператор кадрирования), подкорректировать это решение для работы на DB2 не составит большого труда. Таблицы IT_RESEARCH и IT_APPS задействованы только в этом рецепте, поэтому перед кодом решения приводится код для их создания:

```
create table IT_research (deptno number, ename varchar2(20))
```

```
insert into IT_research values (100, 'HOPKINS')
insert into IT_research values (100, 'JONES')
insert into IT_research values (100, 'TONEY')
insert into IT_research values (200, 'MORALES')
insert into IT_research values (200, 'P.WHITAKER')
insert into IT_research values (200, 'MARCIANO')
insert into IT_research values (200, 'ROBINSON')
insert into IT_research values (300, 'LACY')
insert into IT_research values (300, 'WRIGHT')
insert into IT_research values (300, 'J.TAYLOR')
```

```
create table IT_apps (deptno number, ename varchar2(20))
```

```
insert into IT_apps values (400, 'CORRALES')
insert into IT_apps values (400, 'MAYWEATHER')
insert into IT_apps values (400, 'CASTILLO')
insert into IT_apps values (400, 'MARQUEZ')
insert into IT_apps values (400, 'MOSLEY')
insert into IT_apps values (500, 'GATTI')
```

```
insert into IT_apps values (500,'CALZAGHE')
insert into IT_apps values (600,'LAMOTTA')
insert into IT_apps values (600,'HAGLER')
insert into IT_apps values (600,'HEARNS')
insert into IT_apps values (600,'FRAZIER')
insert into IT_apps values (700,'GUINN')
insert into IT_apps values (700,'JUDAH')
insert into IT_apps values (700,'MARGARITO')

1 select max(decode(flag2,0,it_dept)) research,
2         max(decode(flag2,1,it_dept)) apps
3   from (
4 select sum(flag1)over(partition by flag2
5                   order by flag1,rownum) flag,
6        it_dept, flag2
7   from (
8 select 1 flag1, 0 flag2,
9        decode(rn,1,to_char(deptno),'||ename) it_dept
10  from (
11 select x.*, y.id,
12        row_number()over(partition by x.deptno order by y.id) rn
13   from (
14 select deptno,
15        ename,
16        count(*)over(partition by deptno) cnt
17   from it_research
18        ) x,
19        (select level id from dual connect by level <= 2) y
20        )
21  where rn <= cnt+1
22 union all
23 select 1 flag1, 1 flag2,
24        decode(rn,1,to_char(deptno),'||ename) it_dept
25   from (
26 select x.*, y.id,
27        row_number()over(partition by x.deptno order by y.id) rn
28   from (
29 select deptno,
30        ename,
31        count(*)over(partition by deptno) cnt
32   from it_apps
33        ) x,
34        (select level id from dual connect by level <= 2) y
35        )
36  where rn <= cnt+1
37        ) tmp1
38        ) tmp2
39 group by flag
```


Обсуждение

Подобно многим другим запросам для создания хранилищ данных и/или отчетов, приведенное решение выглядит весьма запутанным. Но если рассмотреть работу его составляющих частей, то можно увидеть, что оно не что иное, как обычное объединение и транспонирование с добавлением декартова произведения. Поэтому сначала мы рассмотрим работу каждой части конструкции UNION ALL, а затем соберем их всех вместе для транспонирования. И начнем с рассмотрения нижней части конструкции UNION ALL:

```
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),''||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
    ) x,
    (select level id from dual connect by level <= 2) y
    ) z
 where rn <= cnt+1
```

FLAG1	FLAG2	IT_DEPT
1	1	400
1	1	MAYWEATHER
1	1	CASTILLO
1	1	MARQUEZ
1	1	MOSLEY
1	1	CORRALES
1	1	500
1	1	CALZAGHE
1	1	GATTI
1	1	600
1	1	HAGLER
1	1	HEARNS
1	1	FRAZIER
1	1	LAMOTTA
1	1	700
1	1	JUDAH
1	1	MARGARITO
1	1	GUINN

Проанализируем, как именно формируется результирующее множество. Разбив предшествующий запрос на его простейшие составляющие, получаем вложенный

запрос X, который просто возвращает из таблицы IT_APPS все ENAME и DEPTNO вместе с количеством служащих в каждом DEPTNO. Далее приводится соответствующий запрос и его результирующее множество:

```
select deptno deptno,
ename,
count(*)over(partition by deptno) cnt
from it_apps
```

DEPTNO	ENAME	CNT
400	CORRALES	5
400	MAYWEATHER	5
400	CASTILLO	5
400	MARQUEZ	5
400	MOSLEY	5
500	GATTI	2
500	CALZAGHE	2
600	LAMOTTA	4
600	HAGLER	4
600	HEARNS	4
600	FRAZIER	4
700	GUINN	3
700	JUDAH	3
700	MARGARITO	3

На следующем шаге создаем декартово произведение между строк, возвращенных вложенным запросом X, и двумя строками, созданными из DUAL функцией CONNECT BY. Далее приводятся соответствующий запрос и его результаты:

```
select *
  from (
select deptno deptno,
      ename,
      count(*)over(partition by deptno) cnt
  from it_apps
    ) x,
      (select level id from dual connect by level <= 2) y
 order by 2
```

DEPTNO	ENAME	CNT	ID
500	CALZAGHE	2	1
500	CALZAGHE	2	2
400	CASTILLO	5	1
400	CASTILLO	5	2
400	CORRALES	5	1
400	CORRALES	5	2
600	FRAZIER	4	1
600	FRAZIER	4	2

```

500 GATTI      2  1
500 GATTI      2  2
700 GUINN     3  1
700 GUINN     3  2
600 HAGLER    4  1
600 HAGLER    4  2
600 HEARNS    4  1
600 HEARNS    4  2
700 JUDAH     3  1
700 JUDAH     3  2
600 LAMOTTA   4  1
600 LAMOTTA   4  2
700 MARGARITO 3  1
700 MARGARITO 3  2
400 MARQUEZ   5  1
400 MARQUEZ   5  2
400 MAYWEATHER 5  1
400 MAYWEATHER 5  2
400 MOSLEY    5  1
400 MOSLEY    5  2

```

Как можно видеть, в результате декартова произведения с вложенным запросом Y каждая строка вложенного запроса X теперь возвращается дважды. Причина необходимости использования декартова произведения вскоре станет ясной. Далее берем текущее результирующее множество и присваиваем каждому служащему ранг в его отделе DEPTNO по номеру ID (который в результате декартова произведения имеет значение 1 или 2). Далее приводится соответствующий запрос ранжирования и его результаты:

```

select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
    ) x,
    (select level id from dual connect by level <= 2) y

```

DEPTNO	ENAME	CNT	ID	RN
400	CORRALES	5	1	1
400	MAYWEATHER	5	1	2
400	CASTILLO	5	1	3
400	MARQUEZ	5	1	4
400	MOSLEY	5	1	5
400	CORRALES	5	2	6
400	MOSLEY	5	2	7
400	MAYWEATHER	5	2	8

400 CASTILLO	5	2	9
400 MARQUEZ	5	2	10
500 GATTI	2	1	1
500 CALZAGHE	2	1	2
500 GATTI	2	2	3
500 CALZAGHE	2	2	4
600 LAMOTTA	4	1	1
600 HAGLER	4	1	2
600 HEARNS	4	1	3
600 FRAZIER	4	1	4
600 LAMOTTA	4	2	5
600 HAGLER	4	2	6
600 FRAZIER	4	2	7
600 HEARNS	4	2	8
700 GUINN	3	1	1
700 JUDAH	3	1	2
700 MARGARITO	3	1	3
700 GUINN	3	2	4
700 JUDAH	3	2	5
700 MARGARITO	3	2	6

Каждому служащему, а затем его дубликату присваивается ранг. Результирующее множество содержит дубликаты записей для всех служащих в таблице IT_APP, а также их ранги в рамках соответствующего отдела DEPTNO. Дубликаты строк создаются потому, что нам требуется место в результирующем множестве для вставки значений DEPTNO в столбец ENAME. Декартово произведение таблицы IT_APPS с однострочной таблицей дополнительных строк не даст, поскольку кардинальность любой таблицы, умноженная на 1, равна кардинальности этой таблицы.

На следующем шаге транспонируем имеющееся на этом этапе результирующее множество таким образом, чтобы все значения ENAME были в одном столбце, но разбиты на группы по отделам с номером отдела DEPT в начале каждой группы. Далее приводится соответствующий запрос и результаты его исполнения:

```
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),''||ename) it_dept
from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
from it_apps
) x,
     (select level id from dual connect by level <= 2) y
) z
where rn <= cnt+1
```

FLAG1	FLAG2	IT_DEPT
1	1	400
1	1	MAYWEATHER
1	1	CASTILLO
1	1	MARQUEZ
1	1	MOSLEY
1	1	CORRALES
1	1	500
1	1	CALZAGHE
1	1	GATTI
1	1	600
1	1	HAGLER
1	1	HEARNS
1	1	FRAZIER
1	1	LAMOTTA
1	1	700
1	1	JUDAH
1	1	MARGARITO
1	1	GUINN

Столбцы `FLAG1` и `FLAG2` будут использованы позже, поэтому пока их можно игнорировать. Сейчас же сконцентрируемся на значениях столбца `IT_DEPT`. Для каждого значения `DEPTNO` возвращается количество строк $CNT \times 2$, но при этом, согласно условию в предикате `WHERE`, нужно всего лишь $CNT+1$ количество строк. Значение `RN` представляет ранг каждого служащего. Возвращаются все строки с рангом меньшим или равным $CNT+1$, т. е. все записи служащих каждого отдела плюс одна дополнительная строка для служащего с первым рангом в своем отделе `DEPTNO`. Эта дополнительная строка предназначена для вывода номера отдела `DEPTNO`. Значение `DEPTNO` можно вставить в результирующее множество с помощью функции `DECODE` (это старая функция Oracle, предоставляющая функциональность более или менее эквивалентную функциональности выражения `CASE`). Служащий в первой позиции (по значению `RN`) по-прежнему включен в результирующее множество в каждом отделе `DEPTNO`, но теперь уже в последней позиции (однако, поскольку порядок не имеет значения, это не проблема). Вот так и работает нижняя часть конструкции `UNION ALL`.

Верхняя часть конструкции `UNION ALL` работает аналогично нижней, поэтому в ее подробном рассмотрении нет надобности. Вместо этого рассмотрим результирующее множество, возвращаемое совмещенными запросами:

```
select 1 flag1, 0 flag2,
       decode(m,1,to_char(deptno),''||ename) it_dept
from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
from (
select deptno,
       ename,
```

```

count(*)over(partition by deptno) cnt
from it_research
  ) x,
  (select level id from dual connect by level <= 2) y
  )
where rn <= cnt+1
union all
select 1 flag1, 1 flag2,
      decode(rn,1,to_char(deptno),' '||ename) it_dept
from (
select x.*, y.id,
      row_number()over(partition by x.deptno order by y.id) rn
from (
select deptno deptno,
      ename,
      count(*)over(partition by deptno) cnt
from it_apps
  ) x,
  (select level id from dual connect by level <= 2) y
  )
where rn <= cnt+1

```

FLAG1	FLAG2	IT_DEPT
1	0	100
1	0	JONES
1	0	TONEY
1	0	HOPKINS
1	0	200
1	0	P.WHITAKER
1	0	MARCIANO
1	0	ROBINSON
1	0	MORALES
1	0	300
1	0	WRIGHT
1	0	J.TAYLOR
1	0	LACY
1	1	400
1	1	MAYWEATHER
1	1	CASTILLO
1	1	MARQUEZ
1	1	MOSLEY
1	1	CORRALES
1	1	500
1	1	CALZAGHE
1	1	GATTI
1	1	600

```

1          1  HAGLER
1          1  HEARNS
1          1  FRAZIER
1          1  LAMOTTA
1          1 700
1          1  JUDAH
1          1  MARGARITO
1          1  GUINN

```

На этом этапе все еще непонятно назначение флага `FLAG1`, но можно видеть, что `FLAG2` указывает, в какой части `UNION ALL` создана строка (0 — в верхней, 1 — в нижней).

Далее нам нужно поместить совмещенное результирующее множество во вложенный запрос и вычислить текущую сумму по `FLAG1` (наконец-то мы узнали его назначение!), которая будет служить рангом для каждой строки в каждой группе. Далее приводятся соответствующий запрос и его результаты:

```

select sum(flag1)over(partition by flag2
                      order by flag1,rownum) flag,
       it_dept, flag2
  from (
select 1 flag1, 0 flag2,
       decode(rn,1,to_char(deptno),' '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_research
   ) x,
   (select level id from dual connect by level <= 2) y
   )
 where rn <= cnt+1
 union all
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),' '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
   ) x,

```

```

    (select level id from dual connect by level <= 2) y
  )
where rn <= cnt+1
) tmp1

```

FLAG	IT_DEPT	FLAG2
1	100	0
2	JONES	0
3	TONEY	0
4	HOPKINS	0
5	200	0
6	P.WHITAKER	0
7	MARCIANO	0
8	ROBINSON	0
9	MORALES	0
10	300	0
11	WRIGHT	0
12	J.TAYLOR	0
13	LACY	0
1	400	1
2	MAYWEATHER	1
3	CASTILLO	1
4	MARQUEZ	1
5	MOSLEY	1
6	CORRALES	1
7	500	1
8	CALZAGHEE	1
9	GATTI	1
10	600	1
11	HAGLER	1
12	HEARNS	1
13	FRAZIER	1
14	LAMOTTA	1
15	700	1
16	JUDAH	1
17	MARGARITO	1
18	GUINN	1

Наконец, в завершение транспонируем по FLAG2 значения, возвращаемые из TMP1, при этом группируя их по FLAG (текущей сумме, создаваемой в TMP1). Результаты из TMP1 помещаются во вложенный запрос и транспонируются (помещенные в конечный вложенный запрос TMP2). Конечный запрос решения и его результирующее множество показаны далее:

```

select max(decode(flag2,0,it_dept)) research,
       max(decode(flag2,1,it_dept)) apps

```



```

from (
select sum(flag1)over(partition by flag2
                      order by flag1,rownum) flag,
       it_dept, flag2
from (
select 1 flag1, 0 flag2,
       decode(rn,1,to_char(deptno),' '||ename) it_dept
from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
from it_research
) x,
(select level id from dual connect by level <= 2) y
)
where rn <= cnt+1
union all
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),' '||ename) it_dept
from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
from it_apps
) x,
(select level id from dual connect by level <= 2) y
)
where rn <= cnt+1
) tmp1
) tmp2
group by flag

```

RESEARCH	APPS
-----	-----
100	400
JONES	MAYWEATHER
TONEY	CASTILLO
HOPKINS	MARQUEZ
200	MOSLEY
P.WHITAKER	CORRALES
MARCIANO	500
ROBINSON	CALZAGHE
MORALES	GATTI

300	600
WRIGHT	HAGLER
J. TAYLOR	HEARNS
LACY	FRAZIER
	LAMOTTA
	700
	JUDAH
	MARGARITO
	GUINN

14.10. Преобразование скалярного подзапроса в составной подзапрос в Oracle

ЗАДАЧА

Нужно обойти ограничение на возвращение скалярным подзапросом только одного значения. Например, вы хотите исполнить следующий запрос:

```
select e.deptno,
       e.ename,
       e.sal,
       (select d.dname,d.loc,sysdate today
        from dept d
        where e.deptno=d.deptno)
from emp e
```

но получаете сообщение об ошибке, т. к. подзапросы в списке `SELECT` могут возвращать только одно значение.

РЕШЕНИЕ

Честно говоря, такую задачу вовсе не стоило бы рассматривать, потому что простое соединение между таблицами `EMP` и `DEPT` позволило бы вам вернуть из таблицы `DEPT` столько значений, сколько вы хотите. Тем не менее цель этого примера — сконцентрироваться на приводимом подходе и понять, как применить его в ситуации, в которой он может оказаться полезным. Ключ к решению задачи обхода требования возврата только одного значения при помещении оператора `SELECT` в другой оператор `SELECT` (скалярный подзапрос) — воспользоваться возможностями объектных типов Oracle. Можно определить объект с несколькими атрибутами, а затем работать с ним как с единичной сущностью или обращаться к каждому элементу индивидуально. Фактически мы вовсе не обходим ограничение, а просто возвращаем одно значение — объект, который содержит множество атрибутов.

В решении используется следующий объектный тип:

```
create type generic_obj
as object (
  val1 varchar2(10),
```

```

    val2 varchar2(10),
    val3 date
);

```

Располагая этим типом, можно выполнить следующий запрос:

```

1 select x.deptno,
2       x.ename,
3       x.multival.val1 dname,
4       x.multival.val2 loc,
5       x.multival.val3 today
6   from (
7select e.deptno,
8       e.ename,
9       e.sal,
10      (select generic_obj(d.dname,d.loc,sysdate+1)
11        from dept d
12        where e.deptno=d.deptno) multival
13   from emp e
14      ) x

```

DEPTNO	ENAME	DNAME	LOC	TODAY
20	SMITH	RESEARCH	DALLAS	12-SEP-2020
30	ALLEN	SALES	CHICAGO	12-SEP-2020
30	WARD	SALES	CHICAGO	12-SEP-2020
20	JONES	RESEARCH	DALLAS	12-SEP-2020
30	MARTIN	SALES	CHICAGO	12-SEP-2020
30	BLAKE	SALES	CHICAGO	12-SEP-2020
10	CLARK	ACCOUNTING	NEW YORK	12-SEP-2020
20	SCOTT	RESEARCH	DALLAS	12-SEP-2020
10	KING	ACCOUNTING	NEW YORK	12-SEP-2020
30	TURNER	SALES	CHICAGO	12-SEP-2020
20	ADAMS	RESEARCH	DALLAS	12-SEP-2020
30	JAMES	SALES	CHICAGO	12-SEP-2020
20	FORD	RESEARCH	DALLAS	12-SEP-2020
10	MILLER	ACCOUNTING	NEW YORK	12-SEP-2020

Обсуждение

Ключ к решению — использовать функцию конструктора объекта (по умолчанию имя функции конструктора такое же, как и имя объекта). Поскольку сам объект является одиночным скалярным значением, он отвечает требованию скалярного подзапроса, как можно видеть в следующем примере:

```

select e.deptno,
       e.ename,
       e.sal,
       (select generic_obj(d.dname,d.loc,sysdate-1)
        from dept d

```

```
where e.deptno=d.deptno) multival
from emp e
```

```
DEPTNO ENAME      SAL MULTIVAL(VAL1, VAL2, VAL3)
-----
20 SMITH          800 GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2020')
30 ALLEN          1600 GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2020')
30 WARD           1250 GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2020')
20 JONES          2975 GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2020')
30 MARTIN         1250 GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2020')
30 BLAKE          2850 GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2020')
10 CLARK          2450 GENERIC_OBJ('ACCOUNTING', 'NEW YORK', '12-SEP-2020')
20 SCOTT          3000 GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2020')
10 KING           5000 GENERIC_OBJ('ACCOUNTING', 'NEW YORK', '12-SEP-2020')
30 TURNER        1500 GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2020')
20 ADAMS          1100 GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2020')
30 JAMES           950 GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2020')
20 FORD           3000 GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2020')
10 MILLER        1300 GENERIC_OBJ('ACCOUNTING', 'NEW YORK', '12-SEP-2020')
```

После этого осталось только поместить запрос во вложенный запрос и извлечь атрибуты.



В отличие от других СУБД, в Oracle обычно не требуется присваивать имена вложенным запросам. Но здесь присвоить имя вложенному запросу необходимо, т. к. в противном случае мы не сможем обращаться к атрибутам объекта.

14.11. Парсинг сериализованных данных в строки

ЗАДАЧА

Требуется выполнить парсинг сериализованных данных (хранящихся в строках) и вернуть их в виде строк таблицы.

Например, пусть у нас есть следующие исходные данные:

```
STRINGS
-----
entry:stewiegriffin:lois:brian:
entry:moe::sizlack:
entry:petergriffin:meg:chris:
entry:willie:
entry:quagmire:mayorwest:cleveland:
entry:::flanders:
entry:robo:tchi:ken:
```

Нам надо преобразовать эти сериализованные строки в следующее результирующее множество:

VAL1	VAL2	VAL3
moe		sizlack
petergriffin	meg	chris
quagmire	mayorwest	cleveland
robo	tchi	ken
stewiegriffin	lois	brian
willie		flanders

РЕШЕНИЕ

В рассматриваемом случае каждая сериализованная строка может хранить до трех значений, разделенных двоеточиями. Строка может содержать и меньше трех значений. Тогда нужно быть особенно внимательным, чтобы поместить имеющиеся значения в правильный столбец результирующего множества. Рассмотрим, например, следующую строку:

```
entry:::flanders:
```

Она представляет запись, содержащую только одно, третье, значение. При этом, посмотрев на результирующее множество в *разд. «Задача»*, мы увидим, что в строке со значением `flanders` значения столбцов `VAL1` и `VAL2` равны `NULL`.

Ключ к решению этой задачи — просто выполнить обход строки, осуществляя в процессе ее парсинг, после чего выполнить простое транспонирование. В решении используются строки из представления `V`, определение которого приводится далее. В примере мы ориентируемся на синтаксис `Oracle`, но, поскольку для его работы не требуется ничего, кроме функций парсинга строк, преобразовать его под другие СУБД не составит большого труда:

```
create view V
  as
select 'entry:stewiegriffin:lois:brian:' strings
  from dual
 union all
select 'entry:moe::sizlack:'
  from dual
 union all
select 'entry:petergriffin:meg:chris:'
  from dual
 union all
select 'entry:willie:'
  from dual
 union all
select 'entry:quagmire:mayorwest:cleveland:'
  from dual
 union all
```

```
select 'entry:::flanders:'
      from dual
      union all
select 'entry:robo:tchi:ken:'
      from dual
```

При использовании представления V в качестве источника строк решение выглядит следующим образом:

```
1 with cartesian as (
2   select level id
3     from dual
4   connect by level <= 100
5 )
6 select max(decode(id,1,substr(strings,p1+1,p2-1))) val1,
7        max(decode(id,2,substr(strings,p1+1,p2-1))) val2,
8        max(decode(id,3,substr(strings,p1+1,p2-1))) val3
9   from (
10  select v.strings,
11         c.id,
12         instr(v.strings,':',1,c.id) p1,
13         instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
14   from v, cartesian c
15  where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
16         )
17  group by strings
18  order by 1
```

Обсуждение

На первом шаге выполняем обход сериализованных строк:

```
with cartesian as (
select level id
      from dual
      connect by level <= 100
)
select v.strings,
       c.id
      from v, cartesian c
      where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
```

STRINGS	ID
entry:::flanders:	1
entry:::flanders:	2
entry:::flanders:	3
entry:moe::sizlack:	1
entry:moe::sizlack:	2

```

entry:moe::sizlack:          3
entry:petergriffin:meg:chris: 1
entry:petergriffin:meg:chris: 3
entry:petergriffin:meg:chris: 2
entry:quagmire:mayorwest:cleveland: 1
entry:quagmire:mayorwest:cleveland: 3
entry:quagmire:mayorwest:cleveland: 2
entry:robo:tchi:ken:        1
entry:robo:tchi:ken:        2
entry:robo:tchi:ken:        3
entry:stewiegriffin:lois:brian: 1
entry:stewiegriffin:lois:brian: 3
entry:stewiegriffin:lois:brian: 2
entry:willie:                1

```

Затем с помощью функции `INSTR` определяем числовую позицию всех двоеточий во всех строках. Так как каждое значение, которое нужно извлечь, заключено в два двоеточия, числовым значениям их позиций присваиваются псевдонимы `P1` и `P2`, означающие соответственно «позиция 1» и «позиция 2»:

```

with cartesian as (
select level id
  from dual
 connect by level <= 100
)
select v.strings,
       c.id,
       instr(v.strings, ':', 1, c.id) p1,
       instr(v.strings, ':', 1, c.id+1)-instr(v.strings, ':', 1, c.id) p2
  from v, cartesian c
 where c.id <= (length(v.strings)-length(replace(v.strings, ':'))) - 1
 order by 1

```

STRINGS	ID	P1	P2
entry::flanders:	1	6	1
entry::flanders:	2	7	1
entry::flanders:	3	8	9
entry:moe::sizlack:	1	6	4
entry:moe::sizlack:	2	10	1
entry:moe::sizlack:	3	11	8
entry:petergriffin:meg:chris:	1	6	13
entry:petergriffin:meg:chris:	3	23	6
entry:petergriffin:meg:chris:	2	19	4
entry:quagmire:mayorwest:cleveland:	1	6	9
entry:quagmire:mayorwest:cleveland:	3	25	10
entry:quagmire:mayorwest:cleveland:	2	15	10
entry:robo:tchi:ken:	1	6	5
entry:robo:tchi:ken:	2	11	5

entry:robo:tchi:ken:	3	16	4
entry:stewiegriffin:lois:brian:	1	6	14
entry:stewiegriffin:lois:brian:	3	25	6
entry:stewiegriffin:lois:brian:	2	20	5
entry:willie:	1	6	7

Теперь, зная числовые позиции каждой пары двоеточий в строках, просто передаем эту информацию функции `SUBSTR`, чтобы извлечь заключенные в них значения. Так как нам нужно создать результирующее множество с тремя столбцами, используем функцию `DECODE` для вычисления значений `ID` из декартова произведения:

```
with cartesian as (
  select level id
  from dual
  connect by level <= 100
)
select decode(id,1,substr(strings,p1+1,p2-1)) val1,
       decode(id,2,substr(strings,p1+1,p2-1)) val2,
       decode(id,3,substr(strings,p1+1,p2-1)) val3
  from (
select v.strings,
       c.id,
       instr(v.strings,':',1,c.id) p1,
       instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
  from v,cartesian c
 where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
  )
 order by 1
```

VAL1	VAL2	VAL3

moe		
petergriffin		
quagmire		
robo		
stewiegriffin		
willie		
	lois	
	meg	
	mayorwest	
	tchi	
		brian
		sizlack
		chris
		cleveland
		flanders
		ken

В завершение применяем агрегатную функцию к значениям, возвращенным функцией SUBSTR, при этом группируя по ID, чтобы придать результирующему множеству удобочитаемый формат:

```
with cartesian as (
select level id
       from dual
       connect by level <= 100
)
select max(decode(id,1,substr(strings,p1+1,p2-1))) val1,
       max(decode(id,2,substr(strings,p1+1,p2-1))) val2,
       max(decode(id,3,substr(strings,p1+1,p2-1))) val3
       from (
select v.strings,
       c.id,
       instr(v.strings,':',1,c.id) p1,
       instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
       from v,cartesian c
       where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
       )
group by strings
order by 1
```

VAL1	VAL2	VAL3
moe		sizlack
petergriffin	meg	chris
quagmire	mayorwest	cleveland
robo	tchi	ken
stewiegriffin	lois	brian
willie		flanders

14.12. Вычисление процентной доли от целого

ЗАДАЧА

Требуется создать отчет с набором числовых значений, при этом каждое значение также нужно показать в виде процентной доли от целого. Например, используя СУБД Oracle, вы хотите вернуть результирующее множество с разбиением зарплат по значению JOB, чтобы определить, какая из должностей обходится компании дороже всего. Результирующее множество также должно содержать количество служащих в каждой должности JOB, чтобы не допустить ошибочного толкования результатов. То есть надо вернуть следующее результирующее множество:

JOB	NUM_EMPS	PCT_OF_ALL_SALARIES
CLERK	4	14
ANALYST	2	20

MANAGER	3	28
SALESMAN	4	19
PRESIDENT	1	17

Как можно видеть, если в отчет не включить количество служащих, то можно подумать, что зарплата президента составляет очень малую долю от общей суммы зарплат. Но, зная, что это зарплата только одного президента, можно получить должное представление, что означают эти 17 процентов.

РЕШЕНИЕ

Подходящие средства для решения этой задачи предоставляет только СУБД Oracle за счет встроенной функции `RATIO_TO_REPORT` (вычислить процентную долю от целого в других СУБД можно с помощью операции деления, как показано в *рецепте 7.11*):

```

1 select job,num_emps,sum(round(pct)) pct_of_all_salaries
2   from (
3 select job,
4        count(*)over(partition by job) num_emps,
5        ratio_to_report(sal)over()*100 pct
6   from emp
7        )
8  group by job,num_emps

```

Обсуждение

На первом шаге с помощью оконной функции `COUNT OVER` возвращаем количество служащих для каждой должности `JOB`. Затем используем функцию `RATIO_TO_REPORT`, чтобы вычислить процентную долю (в виде десятичной дроби) каждой зарплаты от суммы всех зарплат:

```

select job,
       count(*)over(partition by job) num_emps,
       ratio_to_report(sal)over()*100 pct
from emp

```

JOB	NUM_EMPS	PCT
ANALYST	2	10.3359173
ANALYST	2	10.3359173
CLERK	4	2.75624462
CLERK	4	3.78983635
CLERK	4	4.4788975
CLERK	4	3.27304048
MANAGER	3	10.2497847
MANAGER	3	8.44099914
MANAGER	3	9.81912145
PRESIDENT	1	17.2265289
SALESMAN	4	5.51248923
SALESMAN	4	4.30663221

```
SALESMAN      4 5.16795866
SALESMAN      4 4.30663221
```

В завершение с помощью агрегатной функции `SUM` суммируем значения, возвращенные функцией `RATIO_TO_REPORT`, с группированием по `JOB` и `NUM_EMPS`. Чтобы вернуть результаты в виде процентов (т. е. 25, а не 0,25 для 25%), умножаем каждое значение на 100:

```
select job,num_emps,sum(round(pct)) pct_of_all_salaries
      from (
select job,
       count(*)over(partition by job) num_emps,
       ratio_to_report(sal)over()*100 pct
      from emp
     )
  group by job,num_emps
```

JOB	NUM_EMPS	PCT_OF_ALL_SALARIES
CLERK	4	14
ANALYST	2	20
MANAGER	3	28
SALESMAN	4	19
PRESIDENT	1	17

14.13. Проверка на наличие в группе значений определенного значения

ЗАДАЧА

Требуется пометить строку логическим флагом, в зависимости от наличия определенного значения в любой строке группы, в которую входит эта строка. Возьмем, например, ситуацию со студентом, который в течение трех месяцев сдал три экзамена. Успешная сдача одного из этих экзаменов означает успешное прохождение курса, и этот факт отмечается соответствующим флагом. Если же в течение трех месяцев студент не сдал успешно ни одного экзамена, этот факт также помечается соответственным флагом. Рассмотрим следующий пример (данные для него созданы с привлечением средств Oracle. Слегка модифицировав этот код на основе оконных функций, вы сможете исполнять его и в других СУБД):

```
create view V
as
select 1 student_id,
       1 test_id,
       2 grade_id,
       1 period_id,
       to_date('02/01/2020','MM/DD/YYYY') test_date,
       0 pass_fail
from dual union all
```

```

select 1, 2, 2, 1, to_date('03/01/2020','MM/DD/YYYY'), 1 from dual union all
select 1, 3, 2, 1, to_date('04/01/2020','MM/DD/YYYY'), 0 from dual union all
select 1, 4, 2, 2, to_date('05/01/2020','MM/DD/YYYY'), 0 from dual union all
select 1, 5, 2, 2, to_date('06/01/2020','MM/DD/YYYY'), 0 from dual union all
select 1, 6, 2, 2, to_date('07/01/2020','MM/DD/YYYY'), 0 from dual

```

```

select *
  from V

```

STUDENT_ID	TEST_ID	GRADE_ID	PERIOD_ID	TEST_DATE	PASS_FAIL
1	1	2	1	01-FEB-2020	0
1	2	2	1	01-MAR-2020	1
1	3	2	1	01-APR-2020	0
1	4	2	2	01-MAY-2020	0
1	5	2	2	01-JUN-2020	0
1	6	2	2	01-JUL-2020	0

Посмотрев на приведенное результирующее множество, можно увидеть, что наш студент сдавал шесть экзаменов в течение двух трехмесячных периодов. Из них он успешно сдал один экзамен в первом периоде (1 означает успешную сдачу, 0 — неуспешную). Таким образом, требование для первого периода удовлетворено. Но он не сдал успешно ни одного экзамена в течение второго трехмесячного периода, в результате чего значение `PASS_FAIL` для них всех равно 0.

Нам надо создать запрос для обработки этих данных, возвращающий результирующее множество, содержащее, среди прочего, столбец с флагом, указывающим, удовлетворил ли студент требование для рассматриваемого периода. В конечном итоге результирующее множество должно иметь следующий вид:

STUDENT_ID	TEST_ID	GRADE_ID	PERIOD_ID	TEST_DATE	METREQ	IN_PROGRESS
1	1	2	1	01-FEB-2020	+	0
1	2	2	1	01-MAR-2020	+	0
1	3	2	1	01-APR-2020	+	0
1	4	2	2	01-MAY-2020	-	0
1	5	2	2	01-JUN-2020	-	0
1	6	2	2	01-JUL-2020	-	1

Столбец `METREQ` (от англ. *met requirement* — удовлетворил требование) может принимать два значения: + и -, означающие, что студент или удовлетворил, или не удовлетворил, соответственно, требование успешной сдачи как минимум одного экзамена в течение трехмесячного периода. Если студент успешно сдал один экзамен в рассматриваемом периоде, все значения `IN_PROGRESS` за этот период должно быть 0. Если же студент не удовлетворил требование для этого периода, тогда значение `IN_PROGRESS` последнего экзамена должно быть равным 1.

РЕШЕНИЕ

Эта задача представляется весьма сложной, поскольку нам нужно рассматривать строки группы в совокупности, а не по отдельности. Посмотрим на значения столбца `PASS_FAIL` таблицы исходных данных, приведенной в разд. «Задача». Если эти данные обрабатывать построчно, то, предположительно, значение `METREQ` конечного результирующего множества должно быть равным - (знак «минус») для всех строк, за исключением строки, в которой значение `TEST_ID` равно 2. В действительности же это не так. Поэтому необходимо обеспечить обработку строк, как группы. Используя оконную функцию `MAX OVER` можно с легкостью определить, сдал ли успешно студент в течение заданного периода по крайней мере один экзамен. Имея эту информацию, вычислить логические значения для периода можно, просто используя выражения `CASE`:

```

1 select student_id,
2         test_id,
3         grade_id,
4         period_id,
5         test_date,
6         decode( grp_p_f,1,lpad('+',6),lpad('-',6) ) metreq,
7         decode( grp_p_f,1,0,
8               decode( test_date,last_test,1,0 ) ) in_progress
9   from (
10 select V.*,
11        max(pass_fail)over(partition by
12                          student_id,grade_id,period_id) grp_p_f,
13        max(test_date)over(partition by
14                          student_id,grade_id,period_id) last_test
15   from V
16        ) x

```

Обсуждение

Ключ к решению этой задачи — использовать оконную функцию `MAX OVER`, чтобы вернуть наибольшее значение `PASS_FAIL` для каждой группы строк. Поскольку значение `PASS_FAIL` может быть только 1 или 0, то если студент успешно сдал хотя бы один экзамен, функция `MAX OVER` возвратит значение 1 для всей группы. Соответствующий запрос и результаты его исполнения показаны далее:

```

select V.*,
       max(pass_fail)over(partition by
                          student_id,grade_id,period_id) grp_pass_fail
   from V

```

STUDENT_ID	TEST_ID	GRADE_ID	PERIOD_ID	TEST_DATE	PASS_FAIL	GRP_PASS_FAIL
1	1	2	1	01-FEB-2020	0	1
1	2	2	1	01-MAR-2020	1	1
1	3	2	1	01-APR-2020	0	1

1	4	2	2	01-MAY-2020	0	0
1	5	2	2	01-JUN-2020	0	0
1	6	2	2	01-JUL-2020	0	0

Из приведенного результирующего множества можно видеть, что в первом периоде студент успешно сдал по крайней мере один экзамен. Таким образом, значение `GRP_PASS_FAIL` для всей группы строк этого периода будет равно 1. Согласно другому требованию, если за рассматриваемый период студент не сдал успешно ни одного экзамена, флагу `IN_PROGRESS` для последнего экзамена группы присваивается значение 1. Для этого также можно использовать оконную функцию `MAX OVER`:

```
select V.*,
       max(pass_fail)over(partition by
                          student_id,grade_id,period_id) grp_p_f,
       max(test_date)over(partition by
                          student_id,grade_id,period_id) last_test
from V
```

STUDENT_ID	TEST_ID	GRADE_ID	PERIOD_ID	TEST_DATE	PASS_FAIL	GRP_P_F	LAST_TEST
1	1	2	1	01-FEB-2020	0	1	01-APR-2020
1	2	2	1	01-MAR-2020	1	1	01-APR-2020
1	3	2	1	01-APR-2020	0	1	01-APR-2020
1	4	2	2	01-MAY-2020	0	0	01-JUL-2020
1	5	2	2	01-JUN-2020	0	0	01-JUL-2020
1	6	2	2	01-JUL-2020	0	0	01-JUL-2020

Определив период, в котором студент успешно сдал хотя бы один экзамен, и дату последнего экзамена для каждого периода, на последнем шаге просто выполняем некоторое форматирование, чтобы придать результатам удобочитаемый вид. В конечной версии решения для этого — в частности, для создания столбцов `METREQ` и `IN_PROGRESS` — используется функция Oracle `DECODE` (завидуйте лютой завистью, фанаты выражения `CASE`). А для выравнивания значений столбца `METREQ` по правому краю используем функцию `LPAD`:

```
select student_id,
       test_id,
       grade_id,
       period_id,
       test_date,
       decode( grp_p_f,1,lpad('+',6),lpad('-',6) ) metreq,
       decode( grp_p_f,1,0,
              decode( test_date,last_test,1,0 ) ) in_progress
from (
select V.*,
       max(pass_fail)over(partition by
                          student_id,grade_id,period_id) grp_p_f,
       max(test_date)over(partition by
                          student_id,grade_id,period_id) last_test
from V
) x
```

STUDENT_ID	TEST_ID	GRADE_ID	PERIOD_ID	TEST_DATE	METREQ	IN_PROGRESS
1	1	2	1	01-FEB-2020	+	0
1	2	2	1	01-MAR-2020	+	0
1	3	2	1	01-APR-2020	+	0
1	4	2	2	01-MAY-2020	-	0
1	5	2	2	01-JUN-2020	-	0
1	6	2	2	01-JUL-2020	-	1

14.14. Подведем итоги

Язык SQL обладает намного более мощными средствами, чем многие думают. Компонуя материал всей этой книги, мы прилагали усилия, чтобы побудить вас увидеть более обширный, чем заметно на первый взгляд, круг приложений этого языка. В этой же главе мы вообще взяли граничные случаи и попытались показать вам, чего можно достичь с помощью этого языка, используя как его стандартные возможности, так и возможности, специфичные для конкретных СУБД.

Краткий обзор оконных функций

В рецептах этой книги широко используются оконные функции, добавленные в стандарт ISO SQL в 2003 году, а также собственные оконные функции поставщиков СУБД. В этом приложении приводится краткий обзор работы оконных функций. Оконные функции значительно упрощают выполнение многих задач, трудно поддающихся решению посредством стандартных средств SQL. Полный список доступных оконных функций, их синтаксис, а также подробное описание их работы вы найдете в документации на используемую СУБД.

Группировка

Прежде чем приступить к рассмотрению оконных функций, очень важно понимать, как работает в SQL *группировка*, так как приемы группирования результатов в SQL могут быть трудны для понимания. В основе этой проблемы лежит неполное понимание принципов работы оператора `GROUP BY`, а также затруднение с пониманием, почему при использовании этого оператора те или иные запросы возвращают те или иные результаты.

Попросту говоря, группировка — это способ организации подобных строк в одну группу. Суть группировки следующая — каждая строка результирующего множества, возвращенного запросом с использованием оператора `GROUP BY`, является группой и представляет одну или несколько строк с такими же значениями в одном или нескольких указанных столбцах.

Таким образом, практическими примерами групп в таблице EMP являются *все служащие отдела 10* (эти служащие помещаются в одну группу по общему значению `DEPTNO`, равному 10), или *все клерки* (эти служащие помещаются в одну группу по общему значению `JOB = CLERK`). Рассмотрим следующие два запроса. Первый из них возвращает всех служащих в отделе 10, а второй помещает всех служащих отдела 10 в группу и возвращает следующую информацию об этой группе: количество строк (членов) группы и наибольшее и наименьшее значения зарплаты членов группы.

```
select deptno,ename
       from emp
       where deptno=10
```



```
DEPTNO ENAME
```

```
-----
 10 CLARK
 10 KING
 10 MILLER
```

```
select deptno,
       count(*) as cnt,
       max(sal) as hi_sal,
       min(sal) as lo_sal
  from emp
 where deptno=10
 group by deptno
```

```
DEPTNO      CNT      HI_SAL      LO_SAL
-----
 10          3       5000       1300
```

Если бы мы не смогли организовать служащих отдела 10 в одну группу, чтобы получить информацию, предоставленную вторым запросом, нам бы пришлось вручную проанализировать строки для этого отдела. В рассмотренном случае, когда такая группа содержит всего лишь три строки, это не представляет никаких проблем, но вот с тремя миллионами строк ситуация была бы кардинально иной. Итак, какие могут быть причины для группирования записей? Самые разнообразные. Возможно, надо узнать количество разных групп или количество членов (строк) в каждой группе. Как можно видеть из нашего простого примера, группировка позволяет получить информацию о многих строках таблицы без необходимости проверять каждую из них по отдельности.

Определение группы в SQL

В математике в большинстве случаев группа определяется как (G, \bullet, e) , где G представляет множество, \bullet — бинарную операцию над G , а e — члена G . Мы воспользуемся этим определением, чтобы сформулировать базовое определение группы в SQL. Группа SQL определяется как (G, e) , где G — представляет результирующее множество одиночного или автономного запроса, использующего оператор `GROUP BY`, а e — является членом G . При этом удовлетворяются следующие аксиомы:

- ◆ для каждого e в G — e является уникальным (от англ. *distinct* — отдельный, отличный от) и представляет один или несколько экземпляров e ;
- ◆ для каждого e в G — агрегатная функция `COUNT` возвращает значение > 0 .



Результирующее множество включено в определение SQL-группы, чтобы подчеркнуть тот факт, что определение группы делается только для работы с запросами. Таким образом, будет правильно заменить e в каждой аксиоме словом «строка», поскольку, строго говоря, строки в результирующем множестве являются группами.

Поскольку эти свойства являются для групп, как мы их понимаем, фундаментальными, важно доказать их истинность (что мы и сделаем на примере некоторых SQL-запросов).

Группы не могут быть пустыми

По самому своему определению группа должна содержать по крайней мере один член (или строку). Принимая это за истину, можно утверждать, что группу нельзя создать из пустой таблицы. Чтобы доказать истинность этого утверждения, просто попытаемся доказать, что оно ложно. В следующем примере создается пустая таблица, а затем предпринимается попытка создать группы посредством трех разных запросов к этой пустой таблице:

```
create table fruits (name varchar(10))
```

```
select name
   from fruits
  group by name
```

(запрос не возвратил никаких строк)

```
select count(*) as cnt
   from fruits
  group by name
```

(запрос не возвратил никаких строк)

```
select name, count(*) as cnt
   from fruits
  group by name
```

(запрос не возвратил никаких строк)

Как можно видеть из этих запросов, создать из пустой таблицы то, что SQL считает группой, невозможно.

Группы уникальны

Теперь докажем уникальность групп, создаваемых запросами с использованием оператора GROUP BY. В следующем примере в таблицу FRUITS вставляется пять строк, а затем из этих строк создается несколько групп:

```
insert into fruits values ('Oranges')
insert into fruits values ('Oranges')
insert into fruits values ('Oranges')
insert into fruits values ('Apple')
insert into fruits values ('Peach')
```

```
select *
   from fruits
```

```
NAME
-----
Oranges
Oranges
Oranges
Apple
Peach
```

```
select name
      from fruits
      group by name
```

```
NAME
-----
Apple
Oranges
Peach
```

```
select name, count(*) as cnt
      from fruits
      group by name
```

NAME	CNT
Apple	1
Oranges	3
Peach	1

Результаты первого запроса показывают, что таблица FRUITS содержит три строки со значением NAME Oranges. Но второй и третий запросы, в которых используется оператор GROUP BY, возвратили только по одной такой строке. Все эти запросы, вместе взятые, доказывают, что строки в результирующем множестве (e в G , согласно ранее данному определению) являются уникальными, и каждое значение NAME представляет один или несколько экземпляров самого себя в таблице FRUITS.

Важно знать, что группы являются уникальными (distinct), и не использовать ключевое слово DISTINCT в списке SELECT при наличии в запросе оператора GROUP BY.



Мы не утверждаем, что GROUP BY и DISTINCT — это одно и то же. Они представляют две абсолютно разные концепции. Однако надо отметить, что приведенные в списке GROUP BY элементы будут уникальными и в результирующем множестве, и поэтому использовать DISTINCT вместе с GROUP BY просто избыточно.

Аксиома Фреге и парадокс Расселла

Возможно, некоторым читателям будет интересно знать, что аксиома абстрактности Фреге, основанная на определении Кантора членства в бесконечном или неисчислимом множестве, утверждает, что для того или иного определенного идентифицирующего свойства существует множество, чьими членами являются только элементы, обладающие этим свойством. Источником проблемы, согласно Роберту Столлу (Robert Stoll), является «неограниченное использование принципа абстракции». Берtrand Расселл (Bertrand Russell) предложил Готтлобу Фреге (Gottlob Frege) рассмотреть

множество, чьими членами являются множества, определяющим свойством которых является то, что они не являются членами самих себя.

Как подчеркнул Расселл, аксиома абстрактности предоставляет слишком большую свободу, позволяя просто задавать условие или свойство для определения членства во множестве, что создает возможность для противоречий. Чтобы лучше объяснить, как можно обнаружить противоречие, он придумал следующий «парадокс цирюльника»:

В некотором городе есть мужской парикмахер, который бреет всех тех, и только тех, мужчин, которые не бреются сами. Если это действительно так, то кто тогда бреет самого парикмахера?

Рассмотрим более конкретный пример в виде множества, которое можно описать следующим образом:

Все члены x в y , удовлетворяющие определенному условию (P).

Математически это описание выглядит так:

$$\{x \in y \mid P(x)\}$$

Поскольку рассматриваемое множество учитывает только те x в y , которые удовлетворяют условию (P), может быть более понятным описать его так: x является членом y тогда и только тогда, когда x удовлетворяет условию (P).

На этом этапе определим из условия $P(x)$, что x не является членом x :

$$(x \in y)$$

Теперь множество определяется так: x является членом y тогда и только тогда, когда x не является членом x :

$$\{x \in y \mid (x \notin x)\}$$

Возможно, что вам еще не совсем понятна суть парадокса Расселла. Тогда задайте себе такой вопрос: может ли рассматриваемое множество быть членом самого себя? Предположим, что

$$x = y$$

и снова рассмотрим наше множество. Теперь множество можно определить так: y является членом y тогда и только тогда, когда y не является членом y :

$$\{y \in y \mid (y \notin y)\}$$

Простору говоря, парадокс Расселла предполагает существование множества, которое одновременно является и не является членом самого себя, что есть противоречие. Логически рассуждая, можно было бы прийти к выводу, что это вовсе не проблема, — в самом деле, как множество может быть членом самого себя? Ведь, в конце концов, множество всех книг не является одной книгой. Почему же тогда этот парадокс существует и как это может быть проблемой? Проблемой это становится при рассмотрении более абстрактных приложений теории множеств. Например, парадокс Расселла можно продемонстрировать «на практике», рассматривая множество всех множеств. Если разрешить существование такого концепта, тогда в силу самого своего определения оно должно быть членом самого себя (ведь, в конце концов, оно же является множеством всех множеств). Что же тогда будет, если применить предшествующее условие $P(x)$ к множеству всех множеств? Тогда парадокс Расселла предполагал бы, что множество всех множеств является членом самого себя тогда и только тогда, когда оно не является членом самого себя. Очевидное противоречие.

Если кому интересно, Эрнст Цермело (Ernst Zermelo) разработал аксиому схемы выделения (которая также называется *аксиомой схемы подмножеств* или *аксиомой спецификации*), которая элегантно обходит парадокс Рассела в аксиоматической теории множеств.

Значение *COUNT* никогда не равно нулю

Запросы в предыдущем разделе и их результаты также доказывают и вторую аксиому — о том, что агрегатная функция *COUNT* никогда не возвращает ноль при использовании в запросе с *GROUP BY* к непустой таблице. То, что для группы невозможно вернуть нулевой счет, не должно быть удивительным. Мы уже доказали, что группу нельзя создать из пустой таблицы, — следовательно, группа должна иметь по крайней мере одну строку. А если группа содержит по крайней мере одну строку, тогда счет строк всегда будет равен как минимум 1.



Не забывайте, что в рассматриваемом случае речь идет об использовании *COUNT* с *GROUP BY*, а не самостоятельно. Запрос к пустой таблице с использованием *COUNT* без *GROUP BY*, конечно же, возвратит ноль.

Парадоксы

В ответ на открытие Берtrandом Расселлом противоречия в аксиоме абстракции Фреге в теории множеств Готтлоб Фреге (Gottlob Frege) отреагировал следующим образом:

Вряд ли может быть что-либо более прискорбным для автора научного труда, чем разрушение одного из оснований его работы после ее завершения... В такое положение меня поставило получение письма Г-на Бертранда Расселла, когда печать этой работы близилась к завершению.

Парадоксы часто создают сценарии, которые, казалось бы, противоречат установленным теориям или идеям. Во многих случаях эти противоречия локализованы и их можно «обойти», или же они распространяются на такое ничтожное количество тестовых примеров, что их можно безопасно игнорировать.

К этому времени вы, возможно, уже догадались, что суть всех этих рассуждений о парадоксах сводится к тому, что в нашем определении SQL-группы есть парадокс, который нужно разрешить. Хотя в сейчас мы концентрируемся на группах, в конечном итоге мы рассматриваем запросы SQL. В оператор *GROUP BY* запроса может быть включено много разных значений: константы, выражения или, наиболее часто, названия столбцов таблицы. Но за эту гибкость приходится расплачиваться, поскольку в SQL *NULL* является корректным «значением». Значения *NULL* представляют проблему, так как агрегатные функции игнорируют их. С учетом изложенного ранее, если таблица содержит только одну строку со значением *NULL*, что возвратит агрегатная функция *COUNT*, используемая в запросе с оператором *GROUP BY*? Согласно нашему собственному определению, запрос с *GROUP BY* и агрегатной функцией *COUNT* должен вернуть значение большее или равное 1. Тогда что будет в случае наличия значений, которые игнорируются такими функциями, как *COUNT*, и что это означает для нашего определения группы? Рассмотрим следующий пример, который раскрывает парадокс группы *NULL* (используя при необходимости функцию *COALESCE* для удобочитаемости):

```
select *
  from fruits
```

```
NAME
-----
Oranges
Oranges
Oranges
Apple
Peach
```

```
insert into fruits values (null)
insert into fruits values (null)
insert into fruits values (null)
insert into fruits values (null)
insert into fruits values (null)
```

```
select coalesce(name, 'NULL') as name
  from fruits
```

```
NAME
-----
Oranges
Oranges
Oranges
Apple
Peach
NULL
NULL
NULL
NULL
NULL
```

```
select coalesce(name, 'NULL') as name,
       count(name) as cnt
  from fruits
 group by name
```

NAME	CNT
Apple	1
NULL	0
Oranges	3
Peach	1

Казалось бы, что наличие в нашей таблице значений `NULL` вносит противоречие (или парадокс) в наше определение SQL-группы. К счастью, это противоречие не может стать поводом для беспокойства, поскольку оно больше затрагивает реализацию

агрегатных функций, а не наше определение. Рассмотрим последний запрос из приведенного набора примеров. Общую постановку задачи для него можно сформулировать следующим образом:

Подсчитать количество вхождений в таблицу FRUITS каждого названия или подсчитать количество членов в каждой группе.

Проанализировав приведенные ранее выражения INSERT, можно ясно видеть наличие пяти строк со значениями NULL, что означает наличие группы NULL из пяти членов.



Тогда как значение NULL, несомненно, обладает свойствами, отличающими его от других значений, тем не менее оно является значением и оно может составлять группу.

Как же тогда можно создать запрос, возвращающий счет 5, а не 0, и, таким образом, возвращающий требуемую нам информацию, и в то же самое время соответствующий нашему определению группы? Обходное решение парадокса группы значений NULL приводится в следующем примере:

```
select coalesce(name, 'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
```

NAME	CNT
Apple	1
Oranges	3
Peach	1
NULL	5

Чтобы обойти парадокс группы значений NULL, используем функцию COUNT(*), а не COUNT(NAME). Агрегатные функции игнорируют любые значения NULL, имеющиеся в передаваемых им столбцах. Поэтому, чтобы при подсчете строк со значениями NULL функция COUNT не возвращала 0, ей нужно вместо названия столбца передавать звездочку (*). В результате будут подсчитываться строки, а не значения столбца, поэтому наличие или отсутствие в нем значений NULL не будет иметь значения.

Еще один парадокс связан с аксиомой об однозначности каждой группы результирующего множества (для каждого члена e в G). Сущность результирующих множеств и таблиц SQL, которые более точно следует называть *мультимножествами*, или *множествами с повторяющимися элементами* (поскольку они могут содержать дубликаты строк), такова, что делает возможным возвращение результирующего множества с дубликатами групп. Рассмотрим следующие два запроса:

```
select coalesce(name, 'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
 union all
```

```
select coalesce(name, 'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
```

NAME	CNT
-----	-----
Apple	1
Oranges	3
Peach	1
NULL	5
Apple	1
Oranges	3
Peach	1
NULL	5

```
select x.*
  from (
select coalesce(name, 'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
 ) x,
 (select deptno from dept) y
```

NAME	CNT
-----	-----
Apple	1
Apple	1
Apple	1
Apple	1
Oranges	3
Oranges	3
Oranges	3
Oranges	3
Peach	1
Peach	1
Peach	1
Peach	1
NULL	5
NULL	5
NULL	5
NULL	5

Как можно видеть в этих запросах, в конечных результатах группы повторяются. К счастью, по этому поводу можно особо не беспокоиться, поскольку это представляет собой только частичный парадокс. Первое свойство группы утверждает, что для (G, e) G представляет результирующее множество одиночного или автономного

запроса, использующего оператор `GROUP BY`. Попросту говоря, результирующее множество любого запроса, содержащего оператор `GROUP BY`, соответствует нашему определению группы. Дубликаты групп могут возникать лишь при создании мультимножества при сочетании результирующих множеств двух запросов, содержащих оператор `GROUP BY`. Используемый в первом запросе приведенного примера оператор `UNION ALL` выполняет операцию не над множествами, а над мультимножествами, вызывая `GROUP BY` дважды, что фактически равно исполнению двух запросов.



Оператор над множествами `UNION` дубликатов групп не создает.

Во втором запросе приведенного примера выполняется декартово произведение, но для этого нужно сначала создать группу. Таким образом, самодостаточный запрос с оператором `GROUP BY` отвечает нашему определению группы. Ни один из этих двух примеров никоим образом не нарушает определение SQL-группы. Они приведены только для полноты картины, чтобы продемонстрировать, что в SQL возможно практически все.

Взаимосвязь между *SELECT* и *GROUP BY*

Определив и доказав понятие группы, можно переходить к более практическим вопросам, касающимся запросов с использованием оператора `GROUP BY`. При группировании в SQL важно понимать взаимосвязь между операторами `SELECT` и `GROUP BY`. При использовании агрегатных функций, таких как `COUNT`, важно не забывать, что любой элемент в списке `SELECT`, который не является аргументом агрегатной функции, должен быть частью группы. Например, при желании записать выражение `SELECT` в таком виде:

```
select deptno, count(*) as cnt
  from emp
```

необходимо также указать `DEPTNO` в операторе `GROUP BY`:

```
select deptno, count(*) as cnt
  from emp
 group by deptno
```

DEPTNO	CNT
10	3
20	5
30	6

Исключение из этого правила составляют константы и скалярные значения, возвращаемые пользовательскими функциями, оконными функциями и несвязанными скалярными подзапросами. Так как оператор `SELECT` обрабатывается после операто-

ра GROUP BY, эти конструкции допускаются в списке SELECT и их не нужно (а в некоторых случаях и невозможно) указывать в выражении GROUP BY. Например:

```
select 'hello' as msg,
       1 as num,
       deptno,
       (select count(*) from emp) as total,
       count(*) as cnt
from emp
group by deptno
```

MSG	NUM	DEPTNO	TOTAL	CNT
hello	1	10	14	3
hello	1	20	14	5
hello	1	30	14	6

Пусть этот запрос не собьет вас с толку. Элементы в списке SELECT, которых нет в операторе GROUP BY, не меняют ни значения CNT для каждого DEPTNO, ни значения DEPTNO. Основываясь на результатах приведенного запроса, можно дать более точное определение правила о соответствии элементов в списке SELECT и списке GROUP BY при агрегатных операциях:

Элементы в списке SELECT, которые потенциально могут изменить группу или значение, возвращаемое агрегатной функцией, необходимо включать в оператор GROUP BY.

Дополнительные элементы в списке SELECT в приведенном запросе не изменили ни значение CNT любой из групп (для каждого DEPTNO), ни сами группы.

Теперь самое время задать следующий вопрос: какие именно элементы в списке SELECT могут потенциально изменить группу или значение, возвращаемое агрегатной функцией? Ответ простой: названия других столбцов таблицы или таблицы, из которых осуществляется выборка. Рассмотрим возможность добавления в рассматриваемый запрос названия столбца JOB:

```
select deptno, job, count(*) as cnt
from emp
group by deptno, job
```

DEPTNO	JOB	CNT
10	CLERK	1
10	MANAGER	1
10	PRESIDENT	1
20	CLERK	2
20	ANALYST	2
20	MANAGER	1
30	CLERK	1
30	MANAGER	1
30	SALESMAN	4

Добавляя в список еще один столбец таблицы EMP — JOB, мы меняем группу и результирующее множество. Следовательно, теперь в операторе GROUP BY вместе с DEPTNO также нужно указать и столбец JOB, иначе запрос даст сбой. Включение столбца JOB в операторы SELECT и GROUP BY меняет значение запроса из «Сколько служащих в каждом отделе?» на «Сколько разных типов служащих в каждом отделе?». Опять же, обратите внимание на то, что эти группы являются уникальными: значения для DEPTNO и JOB по отдельности не уникальны, но их сочетания (которые указаны в списках GROUP BY и SELECT и, соответственно, в группе) — уникальны (например, сочетание 10 и CLERK встречается только один раз).

Если в списке SELECT указаны только агрегатные функции, тогда в операторе GROUP BY можно указывать любой действительный столбец. Следующие два запроса иллюстрируют этот факт:

```
select count(*)
      from emp
      group by deptno
```

```
COUNT (*)
-----
      3
      5
      6
```

```
select count(*)
      from emp
      group by deptno,job
```

```
COUNT (*)
-----
      1
      1
      1
      2
      2
      1
      1
      1
      4
```

Хотя в списке SELECT не обязательно указывать элементы иные, чем агрегатные функции, часто это делает результаты более понятными и удобными.



Как правило, при использовании оператора GROUP BY и агрегатных функций в нем необходимо указывать любые элементы списка SELECT (из таблицы или таблиц оператора FROM), не используемые в качестве аргументов агрегатной функции. Но MySQL имеет одну «возможность», позволяющую пренебречь этим правилом и указывать в списке SELECT элементы (столбцы таблицы или таблицы, из которых осуществляется выборка), не являющиеся аргументами агрегатной функции и отсутствующие в операторе GROUP BY. Слово «возможность» заключено в кавычки по-

тому, что полезность ее весьма сомнительна и ее использование чревато возникновением ошибок. В действительности, мы рекомендуем пользователям MySQL, которым хоть сколько-нибудь важна корректность исполнения создаваемых ими запросов, не использовать эту так называемую «возможность».

Оконные функции

Если вы разобрались с понятием группировки и использованием агрегатных операций в SQL, то понять, что такое «оконные функции», вам будет совсем просто. Подобно агрегатным функциям, оконные функции осуществляют операцию агрегации по заданному множеству (группу) строк, но вместо возвращения одного значения на группу могут возвращать несколько значений для каждой группы. Группа строк, над которыми выполняется операция агрегации, называется *окном*. В DB2 такие функции называются *функциями OLAP* (Online Analytic Processing, оперативный анализ данных), в Oracle — *аналитическими функциями*. Но в стандарте ISO SQL они называются *оконными функциями*, поэтому в нашей книге мы используем именно этот термин.

Простой пример

Предположим, что нам нужно подсчитать общее количество служащих во всех отделах. Обычно для этого используется запрос `COUNT(*)` ко всей таблице EMP:

```
select count(*) as cnt
      from emp
```

```
CNT
-----
14
```

Это довольно легко, но часто нам нужно извлечь из строк такие агрегатные данные, которые не являются группировкой или представляют другую группировку.

Благодаря оконным функциям решение таких задач не представляет никаких сложностей. Например, следующий запрос демонстрирует использование оконной функции для получения агрегатных данных (общее число служащих) из заданных строк (по одной на каждого служащего):

```
select ename,
       deptno,
       count(*) over() as cnt
      from emp
     order by 2
```

ENAME	DEPTNO	CNT
CLARK	10	14
KING	10	14
MILLER	10	14
SMITH	20	14

ADAMS	20	14
FORD	20	14
SCOTT	20	14
JONES	20	14
ALLEN	30	14
BLAKE	30	14
MARTIN	30	14
JAMES	30	14
TURNER	30	14
WARD	30	14

В этом примере оконной функцией является `COUNT(*) OVER()`. Ключевое слово `OVER` означает, что эта функция будет вызываться как оконная, а не агрегатная. То есть стандарт SQL разрешает использовать все агрегатные функции как оконные, добавляя к ним в конце ключевое слово `OVER`.

Что же именно делает оконная функция `COUNT(*) OVER()`? Для каждой возвращенной записью строки она возвращает количество *всех строк* в таблице. Как можно предположить по его пустым скобкам, ключевое слово `OVER` принимает дополнительные операторы, позволяющие определять диапазон строк, обрабатываемых этой оконной функцией. При отсутствии таких операторов оконная функция обрабатывает все строки результирующего множества, чем и объясняется значение 14, повторяющееся в каждой строке результата.

Надеемся, вы начинаете понимать огромную полезность оконных функций, которая заключается в том, что они позволяют работать с несколькими уровнями агрегации в одной строке. Продолжая изучать материал этого приложения, вы еще сильнее поймете, насколько невероятно полезной может быть эта способность.

Порядок обработки

Прежде чем углубиться в рассмотрение оператора `OVER`, важно отметить, что оконные функции выполняются как последний шаг обработки SQL перед оператором `ORDER BY`. Рассмотрим, как оконные функции обрабатываются в последнюю очередь, на примере запроса из предыдущего раздела, используя в нем предикат `WHERE`, чтобы отфильтровать служащих отделов 20 и 30:

```
select ename,
       deptno,
       count(*) over() as cnt
from emp
where deptno = 10
order by 2
```

ENAME	DEPTNO	CNT
CLARK	10	3
KING	10	3
MILLER	10	3

Теперь значение `CNT` для каждой строки здесь уже не 14, а 3. В этом примере предикат `WHERE` ограничивает количество строк результирующего множества до трех, поэтому оконная функция подсчитывает только три строки (к тому моменту, когда обработка запроса доходит до части `SELECT` запроса, для оконной функции остаются доступными только три строки). Таким образом, этот пример подтверждает, что оконные функции выполняются после обработки таких операторов, как `WHERE` и `GROUP BY`.

Сегменты

Сегмент или группа строк, над которыми выполняется агрегация, определяется с помощью оператора `PARTITION BY`. Как мы уже видели, при пустых скобках операция агрегации оконной функции распространяется на сегмент, являющийся всем результирующим множеством. Оператор `PARTITION BY` можно рассматривать как «скользящий `GROUP BY`», поскольку, в отличие от обычного `GROUP BY`, группы, создаваемые `PARTITION BY`, не являются уникальными в результирующем множестве. С помощью оператора `PARTITION BY` можно вычислять агрегат заданной группы строк (начиная отсчет заново для каждой новой группы), возвращая каждое значение (каждого члена группы), а не представляя одной группой все экземпляры этого значения в таблице. Рассмотрим следующий запрос:

```
select ename,
       deptno,
       count(*) over(partition by deptno) as cnt
from emp
order by 2
```

ENAME	DEPTNO	CNT
CLARK	10	3
KING	10	3
MILLER	10	3
SMITH	20	5
ADAMS	20	5
FORD	20	5
SCOTT	20	5
JONES	20	5
ALLEN	30	6
BLAKE	30	6
MARTIN	30	6
JAMES	30	6
TURNER	30	6
WARD	30	6

Этот запрос также возвращает 14 строк, но теперь из-за использования выражения `PARTITION BY DEPTNO` функция `COUNT` подсчитывает значение `CNT` для каждого отдела. Значения `CNT` для всех служащих одного отдела (одного сегмента) будут одинаково-

выми, поскольку вычисление агрегата начинается заново только при обнаружении нового отдела. Обратите внимание и на то, что вместе с членами каждой группы также возвращается и информация об этой группе. Предшествующий запрос можно рассматривать как более эффективную версию следующего запроса:

```
select e.ename,
       e.deptno,
       (select count(*) from emp d
        where e.deptno=d.deptno) as cnt
from emp e
order by 2
```

ENAME	DEPTNO	CNT
CLARK	10	3
KING	10	3
MILLER	10	3
SMITH	20	5
ADAMS	20	5
FORD	20	5
SCOTT	20	5
JONES	20	5
ALLEN	30	6
BLAKE	30	6
MARTIN	30	6
JAMES	30	6
TURNER	30	6
WARD	30	6

Еще одно полезное свойство оператора `PARTITION BY` состоит в том, что он выполняет вычисления независимо от других оконных функций, осуществляя сегментирование по другим столбцам в том же выражении `SELECT`. Рассмотрим следующий запрос, который возвращает всех служащих, их отделы, количество служащих в каждом отделе, их должности и количество служащих в каждой должности:

```
select ename,
       deptno,
       count(*) over(partition by deptno) as dept_cnt,
       job,
       count(*) over(partition by job) as job_cnt
from emp
order by 2
```

ENAME	DEPTNO	DEPT_CNT	JOB	JOB_CNT
MILLER	10	3	CLERK	4
CLARK	10	3	MANAGER	3
KING	10	3	PRESIDENT	1
SCOTT	20	5	ANALYST	2

Так как в запросе используется оператор `COUNT(*)`, функция подсчитывает строки. Можно видеть, что 10 служащих комиссионных не получают (те, для кого возвращено значение `-1`). Но если в функцию `COUNT` вместо `*` передать `COMM`, то результаты будут значительно иными:

```
select coalesce(comm,-1) as comm,
       count(comm)over(partition by comm) as cnt
from emp
```

COMM	CNT
0	1
300	1
500	1
1400	1
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0

Так как в этом запросе используется `COUNT(COMM)`, то для столбца `COMM` подсчитываются только не-NULL значения. То есть один служащий получает комиссионные размером 0, один — размером 300 и так далее. Но обратите внимание на значения `CNT` служащих, не получающих комиссионных (со значением `COMM`, равным `NULL`). Эти значения равны 0. Почему? Потому что агрегатные функции игнорируют значения `NULL`, или, более точно, агрегатные функции подсчитывают только не-NULL значения.



При использовании функции `COUNT` подумайте, нужно ли подсчитывать значения `NULL`. Если нет, тогда используйте формат `COUNT(имя_столбца)`. В противном случае используйте формат `COUNT(*)`, в результате чего будут подсчитываться не значения столбцов, а строки.

Когда порядок имеет значение

Иногда порядок обработки строк оконными функциями оказывает существенное влияние на возвращаемые запросом результаты. По этой причине синтаксис оконных функций содержит оператор `ORDER BY`, который можно поместить в оператор `OVER`. Оператор `ORDER BY` используется для указания способа упорядочивания строк в сегменте (не забываем, что при отсутствии оператора `PARTITION BY` «сегмент» означает все результирующее множество).



Некоторые оконные функции требуют упорядочивания сегментов обрабатываемых строк. Вследствие этого для таких оконных функций использование оператора `ORDER BY` является обязательным. На момент подготовки материала этой книги SQL Server не допускает `ORDER BY` в операторе `OVER` при работе с агрегатными оконными функциями. Однако при использовании с ранжирующими оконными функциями присутствие `ORDER BY` в операторе `OVER` допускается.

Оператор `ORDER BY`, используемый в операторе `OVER` оконной функции, задает следующие два параметра:

- ◆ порядок расположения строк в сегменте;
- ◆ строки, включаемые в вычисление.

Рассмотрим следующий запрос, который суммирует и вычисляет текущую сумму зарплат служащих отдела 10:

```
select deptno,
       ename,
       hiredate,
       sal,
       sum(sal)over(partition by deptno) as total1,
       sum(sal)over() as total2,
       sum(sal)over(order by hiredate) as running_total
from emp
where deptno=10
```

DEPTNO	ENAME	HIREDATE	SAL	TOTAL1	TOTAL2	RUNNING_TOTAL
10	CLARK	09-JUN-1981	2450	8750	8750	2450
10	KING	17-NOV-1981	5000	8750	8750	7450
10	MILLER	23-JAN-1982	1300	8750	8750	8750



Чтобы вы не расслабились, в запрос включено суммирование с пустыми скобками. Обратите внимание на то, что значения столбцов `TOTAL1` и `TOTAL2` одинаковые. Почему? Опять же, ответ лежит в плоскости порядка обработки оконных функций. Оператор `WHERE` фильтрует результирующее множество таким образом, что для суммирования берутся только зарплаты отдела 10 (`DEPTNO 10`). В нашем случае есть только один сегмент — все результирующее множество, состоящее только из строк с зарплатами отдела 10. Поэтому значения `TOTAL1` и `TOTAL2` одинаковые.

По значениям столбца `SAL` можно легко видеть, откуда взялись значения столбца `RUNNING_TOTAL`. Можете сами прикидочно сложить их, чтобы получить текущую сумму. Но что более важно, так это вообще понять, каким образом включение `ORDER BY` в оператор `OVER` обеспечило вычисление текущей суммы. Дело в том, что использование `ORDER BY` в операторе `OVER` обуславливает задание «движущегося», или «скользящего», окна по умолчанию — даже несмотря на то, что мы его не видим. Оператор `ORDER BY HIREDATE` завершает суммирование на значении `HIREDATE` в текущей строке.

Следующий запрос такой же, как и предыдущий, но в нем используется оператор `RANGE BETWEEN` (который мы рассмотрим более подробно чуть далее), чтобы явно задать поведение по умолчанию в результате применения `ORDER BY HIREDATE`:

```

select deptno,
       ename,
       hiredate,
       sal,
       sum(sal)over(partition by deptno) as total1,
       sum(sal)over() as total2,
       sum(sal)over(order by hiredate
                    range between unbounded preceding
                    and current row) as running_total
from emp
where deptno=10

```

DEPTNO	ENAME	HIREDATE	SAL	TOTAL1	TOTAL2	RUNNING_TOTAL
10	CLARK	09-JUN-1981	2450	8750	8750	2450
10	KING	17-NOV-1981	5000	8750	8750	7450
10	MILLER	23-JAN-1982	1300	8750	8750	8750

Согласно терминологии ANSI, оператор `RANGE BETWEEN`, используемый в этом запросе, называется *оператором кадрирования* (*framing clause*), и мы будем придерживаться этой терминологии. Теперь должно быть ясно, почему задание `ORDER BY` в операторе `OVER` вычисляет текущую сумму: запросу было указано (по умолчанию) суммировать все строки, начиная с текущей строки и включая все предшествующие строки («предшествующие» согласно определению в `ORDER BY` — в нашем случае упорядочивая строки по `HIREDATE`).

Оператор кадрирования

Давайте применим к результирующему множеству предшествующего запроса оператор кадрирования, начиная с первого принятого на работу служащего — `CLARK`:

1. Вычисляем общую сумму, начиная с зарплаты служащего `CLARK` (2450) и включая всех служащих, принятых на работу до него. Так как `CLARK` был первым служащим, принятым на работу в отделе 10, общая сумма просто равно его зарплате — 2450. Это первое значение, возвращенное в столбце `RUNNING_TOTAL`.
2. Переходим к служащему со следующим `HIREDATE` — `KING` и опять применяем оператор кадрирования. Вычисляем сумму по `SAL`, начиная со значения текущей строки (5000 — зарплата `KING`) и включая значения всех предыдущих строк (всех служащих, нанятых до `KING`). До служащего `KING` был принят только `CLARK`, поэтому текущая сумма будет 5000 + 2450, или 7450. Это второе значение, возвращенное в столбце `RUNNING_TOTAL`.
3. Переходим к `MILLER` — последнему служащему в сегменте на основе `HIREDATE` и снова применяем оператор кадрирования. Вычисляем сумму по `SAL`, начиная со значения текущей строки (1300 — зарплата `MILLER`) и включая значения всех предыдущих строк (всех служащих, нанятых до `MILLER`). До `MILLER` были наняты `CLARK` и `KING`, поэтому их зарплаты включены в значение `RUNNING_TOTAL` строки для

MILLER: $2450 + 5000 + 13000 = 8750$. Это и есть значение `RUNNING_TOTAL` строки для MILLER.

Как можно видеть, текущая сумма вычисляется, по сути, оператором кадрирования. Оператор `ORDER BY` определяет порядок вычислений, а также подразумевает кадрирование по умолчанию.

В целом оператор кадрирования позволяет определять разные «подокна» данных, которые нужно включить в вычисления. Такие подокна можно задавать разными способами. Рассмотрим следующий запрос:

```
select deptno,
       ename,
       sal,
       sum(sal) over (order by hiredate
                    range between unbounded preceding
                           and current row) as run_total1,
       sum(sal) over (order by hiredate
                    rows between 1 preceding
                           and current row) as run_total2,
       sum(sal) over (order by hiredate
                    range between current row
                           and unbounded following) as run_total3,
       sum(sal) over (order by hiredate
                    rows between current row
                           and 1 following) as run_total4
from emp
where deptno=10
```

DEPTNO	ENAME	SAL	RUN_TOTAL1	RUN_TOTAL2	RUN_TOTAL3	RUN_TOTAL4
10	CLARK	2450	2450	2450	8750	7450
10	KING	5000	7450	7450	6300	6300
10	MILLER	1300	8750	6300	1300	1300

Не пугайтесь, этот запрос не такой сложный, каким он выглядит. В предыдущем примере мы уже рассмотрели получение текущей суммы `RUN_TOTAL1` и результаты работы выражения кадрирования `UNBOUNDED PRECEDING AND CURRENT ROW`. Далее приводится краткое описание происходящего в примерах других вычислений.

Текущая сумма `RUN_TOTAL2`

Вместо ключевого слова `RANGE` в этом операторе кадрирования указывается `ROWS`, означая, что *кадр*, или окно, будет формироваться посредством подсчета некоторого количества строк. Выражение `1 PRECEDING` означает, что началом кадра будет строка, непосредственно предшествующая текущей строке. Диапазон охватывает строки вплоть до `CURRENT ROW`, включая ее. Таким образом, в `RUN_TOTAL2` отображается сумма зарплат текущего служащего и служащего с предшествующей датой приема на работу `HIREDATE`.



Так получается, что для CLARK и KING значения RUN_TOTAL1 и RUN_TOTAL2 одинаковые. Почему? Подумайте, какие значения суммируются для каждого из этих служащих в каждой из двух оконных функций. Вдумайтесь, и вы найдете ответ.

Текущая сумма RUN_TOTAL3

Оконная функция для RUN_TOTAL3 работает противоположным образом, чем функция для RUN_TOTAL1. В частности, вместо добавления к значению текущей строки значений всех предшествующих строк добавляются значения всех последующих строк.

Текущая сумма RUN_TOTAL4

А это инверсия функции для RUN_TOTAL2. В частности, вместо добавления к значению текущей строки значения одной предшествующей строки добавляется значение одно последующей строки.



Если вы понимаете все, что было изложено до сих пор, у вас не должно возникнуть проблем с пониманием любых рецептов этой книги. Но если же у вас все же возникают проблемы с пониманием этого материала, попробуйте попрактиковаться на своих данных и примерах. Обычно новые возможности легче понять, создавая код с ними, чем просто читая о них.

Завершаем рассмотрение вопроса кадрирования

В качестве последнего примера воздействия оператора кадрирования на результаты запроса рассмотрим следующий запрос:

```
select ename,
       sal,
       min(sal)over(order by sal) min1,
       max(sal)over(order by sal) max1,
       min(sal)over(order by sal
                    range between unbounded preceding
                    and unbounded following) min2,
       max(sal)over(order by sal
                    range between unbounded preceding
                    and unbounded following) max2,
       min(sal)over(order by sal
                    range between current row
                    and current row) min3,
       max(sal)over(order by sal
                    range between current row
                    and current row) max3,
       max(sal)over(order by sal
                    rows between 3 preceding
                    and 3 following) max4
from emp
```

ENAME	SAL	MIN1	MAX1	MIN2	MAX2	MIN3	MAX3	MAX4
SMITH	800	800	800	800	5000	800	800	1250
JAMES	950	800	950	800	5000	950	950	1250
ADAMS	1100	800	1100	800	5000	1100	1100	1300
WARD	1250	800	1250	800	5000	1250	1250	1500
MARTIN	1250	800	1250	800	5000	1250	1250	1600
MILLER	1300	800	1300	800	5000	1300	1300	2450
TURNER	1500	800	1500	800	5000	1500	1500	2850
ALLEN	1600	800	1600	800	5000	1600	1600	2975
CLARK	2450	800	2450	800	5000	2450	2450	3000
BLAKE	2850	800	2850	800	5000	2850	2850	3000
JONES	2975	800	2975	800	5000	2975	2975	5000
SCOTT	3000	800	3000	800	5000	3000	3000	5000
FORD	3000	800	3000	800	5000	3000	3000	5000
KING	5000	800	5000	800	5000	5000	5000	5000

Разложим этот запрос на составляющие.

Столбец *MIN1*

Оконная функция, которая создает этот столбец, не содержит оператора кадрирования, поэтому задействуется оператор кадрирования по умолчанию `UNBOUNDED PRECEDING AND CURRENT ROW`. Почему значение этого столбца равно 800 для всех строк? Потому, что сначала учитывается минимальная зарплата (`ORDER BY SAL`), которая так и продолжает оставаться минимальной.

Столбец *MAX1*

Значения столбца `MAX1` намного отличаются от значений столбца `MIN1`. Почему? Ответом, опять же, является оператор кадрирования по умолчанию `UNBOUNDED PRECEDING AND CURRENT ROW`. В сочетании с `ORDER BY SAL` он обеспечивает то, что максимальная зарплата также будет соответствовать значению зарплаты в текущей строке.

Рассмотрим первую строку — для служащего `SMITH`. При вычислении суммы зарплаты `SMITH` и зарплат всех предшествующих служащих значением `MAX1` для `SMITH` будет значение его же зарплаты — ввиду отсутствия предшествующих служащих. Далее сравниваем зарплату служащего в следующей строке (`JAMES`) с зарплатами всех предшествующих служащих — то есть просто с зарплатой служащего `SMITH`. Зарплата `JAMES` больше зарплаты `SMITH` и, следовательно, будет максимальной. Применяя эту логику ко всем строкам, увидим, что значением `MAX1` для каждой строки является значение зарплаты текущего служащего.

Столбцы *MIN2* и *MAX2*

Оператором кадрирования для этих столбцов является `UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`, что равнозначно пустым скобкам. Таким образом, при вычисле-

нии `MIN` и `MAX` рассматриваются все строки результирующего множества. Как и следовало ожидать, значения `MIN` и `MAX` для всего результирующего множества постоянны и, следовательно, постоянными являются и значения этих столбцов.

Столбцы `MIN3` и `MAX3`

Для этих столбцов оператором кадрирования является `CURRENT ROW AND CURRENT ROW`, что просто означает принятие к рассмотрению только зарплаты текущего служащего при поиске значений зарплаты `MIN` и `MAX`. Таким образом, значения `MIN3` и `MAX3` для каждой строки такие же, как и значение `SAL`. В этом не было ничего трудного, не так ли?

Столбец `MAX4`

Для этого столбца оператором кадрирования является `3 PRECEDING AND 3 FOLLOWING`. Это означает, что для каждой строки рассматриваются три строки предшествующие и три строки следующие за текущей строкой, а также сама текущая строка. Такой вызов функции `MAX(SAL)` возвратит из этих строк наибольшее значение зарплаты.

Каким образом применяется оператор кадрирования можно понять по значению `MAX4` для служащего `MARTIN`. Зарплата этого служащего составляет 1250, а зарплаты трех служащих, предшествующих ему, составляют: 1250 (`WARD`), 1100 (`ADAMS`) и 950 (`JAMES`). Зарплаты трех служащих, следующих за `MARTIN`, составляют: 1300 (`MILLER`), 1500 (`TURNER`) и 1600 (`ALLEN`). Из всех этих зарплат, включая зарплату самого `MARTIN`, максимальной является зарплата для `ALLEN`, и, следовательно, значение `MAX4` для `MARTIN` равно 1600.

Удобочитаемость + производительность = мощь

Как можно видеть, оконные функции обладают чрезвычайной мощью, поскольку позволяют создавать запросы, содержащие как детальную, так и общую информацию. Благодаря им можно составлять более компактные и эффективные запросы по сравнению с использованием самообъединений и/или скалярных подзапросов. Рассмотрим запрос, который с легкостью предоставляет нам всю следующую информацию:

- ◆ количество служащих в каждом отделе;
- ◆ количество разных должностей в каждом отделе (например, количество клерков в отделе 10);
- ◆ общее количество служащих в таблице `EMP`.

```
select deptno,
       job,
       count(*) over (partition by deptno) as emp_cnt,
       count(job) over (partition by deptno,job) as job_cnt,
       count(*) over () as total
from emp
```

DEPTNO	JOB	EMP_CNT	JOB_CNT	TOTAL
10	CLERK	3	1	14
10	MANAGER	3	1	14
10	PRESIDENT	3	1	14
20	ANALYST	5	2	14
20	ANALYST	5	2	14
20	CLERK	5	2	14
20	CLERK	5	2	14
20	MANAGER	5	1	14
30	CLERK	6	1	14
30	MANAGER	6	1	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14

Чтобы получить такую же информацию без использования оконных функций, потребовалось бы выполнить несколько больший объем работы:

```
select a.deptno, a.job,
       (select count(*) from emp b
        where b.deptno = a.deptno) as emp_cnt,
       (select count(*) from emp b
        where b.deptno = a.deptno and b.job = a.job) as job_cnt,
       (select count(*) from emp) as total
from emp a
order by 1,2
```

DEPTNO	JOB	EMP_CNT	JOB_CNT	TOTAL
10	CLERK	3	1	14
10	MANAGER	3	1	14
10	PRESIDENT	3	1	14
20	ANALYST	5	2	14
20	ANALYST	5	2	14
20	CLERK	5	2	14
20	CLERK	5	2	14
20	MANAGER	5	1	14
30	CLERK	6	1	14
30	MANAGER	6	1	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14

Очевидно, что создать решение без использования оконных функций нетрудно, но оно, определенно, будет не столь совершенным или эффективным. Разницу в производительности при работе с 14-строчной таблицей вы не ощутите, но, применив

эти запросы к таблице в 1000 или 10 000 строк, можно будет увидеть преимущество использования оконных функций над многократным самообъединением или скалярными подзапросами.

Запросы «в стиле отчетов»

Кроме удобочитаемости и производительности, оконные функции полезны тем, что предоставляют «основу» для более сложных запросов «в стиле отчетов». Рассмотрим, например, следующий запрос «в стиле отчета», в котором во вложенном запросе используются оконные функции, а результаты затем агрегируются во внешнем запросе. Оконные функции позволяют возвращать как подробные, так и общие данные, что делает их полезными для составления отчетов. В следующем запросе оконные функции используются для подсчета значений в разных сегментах. Так как агрегатные значения вычисляются для многих строк, вложенный запрос возвращает все строки таблицы EMP, которые транспонируются внешними выражениями CASE для создания отформатированного отчета:

```
select deptno,
       emp_cnt as dept_total,
       total,
       max(case when job = 'CLERK'
                then job_cnt else 0 end) as clerks,
       max(case when job = 'MANAGER'
                then job_cnt else 0 end) as mgrs,
       max(case when job = 'PRESIDENT'
                then job_cnt else 0 end) as prez,
       max(case when job = 'ANALYST'
                then job_cnt else 0 end) as anals,
       max(case when job = 'SALESMAN'
                then job_cnt else 0 end) as smen
  from (
select deptno,
       job,
       count(*) over (partition by deptno) as emp_cnt,
       count(job) over (partition by deptno,job) as job_cnt,
       count(*) over () as total
  from emp
 ) x
 group by deptno, emp_cnt, total
```

DEPTNO	DEPT_TOTAL	TOTAL	CLERKS	MGRS	PREZ	ANALS	SMEN
10	3	14	1	1	1	0	0
20	5	14	2	1	0	2	0
30	6	14	1	1	0	0	4

Приведенный запрос возвращает номера всех отделов, количество служащих в каждом отделе, общее количество служащих в таблице EMP, а также количество

разных должностей в каждом отделе. И вся эта информация предоставляется посредством одного запроса, без использования дополнительных объединений или временных таблиц!

В качестве последнего примера, поясняющего, как с помощью оконных функций можно с легкостью дать ответы на несколько вопросов, рассмотрим следующий запрос:

```
select ename as name,
       sal,
       max(sal)over(partition by deptno) as hiDpt,
       min(sal)over(partition by deptno) as loDpt,
       max(sal)over(partition by job) as hiJob,
       min(sal)over(partition by job) as loJob,
       max(sal)over() as hi,
       min(sal)over() as lo,
       sum(sal)over(partition by deptno
                   order by sal,empno) as dptRT,
       sum(sal)over(partition by deptno) as dptSum,
       sum(sal)over() as ttl
from emp
order by deptno,dptRT
```

NAME	SAL	HIDPT	LODPT	HIJOB	LOJOB	HI	LO	DPTRT	DPTSUM	TTL
MILLER	1300	5000	1300	1300	800	5000	800	1300	8750	29025
CLARK	2450	5000	1300	2975	2450	5000	800	3750	8750	29025
KING	5000	5000	1300	5000	5000	5000	800	8750	8750	29025
SMITH	800	3000	800	1300	800	5000	800	800	10875	29025
ADAMS	1100	3000	800	1300	800	5000	800	1900	10875	29025
JONES	2975	3000	800	2975	2450	5000	800	4875	10875	29025
SCOTT	3000	3000	800	3000	3000	5000	800	7875	10875	29025
FORD	3000	3000	800	3000	3000	5000	800	10875	10875	29025
JAMES	950	2850	950	1300	800	5000	800	950	9400	29025
WARD	1250	2850	950	1600	1250	5000	800	2200	9400	29025
MARTIN	1250	2850	950	1600	1250	5000	800	3450	9400	29025
TURNER	1500	2850	950	1600	1250	5000	800	4950	9400	29025
ALLEN	1600	2850	950	1600	1250	5000	800	6550	9400	29025
BLAKE	2850	2850	950	2975	2450	5000	800	9400	9400	29025

Этот запрос легко и эффективно (и без использования дополнительных объединений с таблицей EMP) предоставляет следующую информацию в удобочитаемом формате. Просто сопоставляем имя служащего и его зарплату с разными строками результирующего множества, чтобы узнать:

- ◆ кто среди всех служащих получает наибольшую зарплату (HI);
- ◆ кто среди всех служащих получает наименьшую зарплату (LO);
- ◆ кто получает наибольшую зарплату в своем отделе (HIDPT);

- ◆ кто получает наименьшую зарплату в своем отделе (LODPT);
- ◆ кто получает наибольшую зарплату для своей должности (HIJOB);
- ◆ кто получает наименьшую зарплату для своей должности (LOJOB);
- ◆ общую сумму всех зарплат (TTL);
- ◆ общую сумму всех зарплат для заданного отдела (DPTSUM);
- ◆ текущую сумму всех зарплат для каждого отдела (DPTRT).

Подзапросы и обобщенные табличные выражения

То, что осуществляют многие из рассмотренных в этой книге запросов, выходит за пределы возможного при использовании таблиц в том виде, в котором они обычно доступны в базах данных. Это особенно относится к агрегатным и оконным функциям. Поэтому для некоторых запросов необходимо создавать производные таблицы — или подзапросы, или обобщенные табличные выражения (ОТВ).

Подзапросы

Пожалуй, для создания виртуальной таблицы, позволяющей исполнять запросы с оконными или агрегатными функциями, будет проще всего использовать подзапрос. Для этого нужно просто поместить требуемый запрос в скобки, а затем создать другой запрос, который будет его использовать. В следующем запросе демонстрируется использование подзапросов с простой двойной агрегацией: кроме количества служащих в каждой должности, нам также нужно вернуть и должность с наибольшим количеством служащих, однако стандартный запрос не позволяет прямое вложение агрегатных функций.

Здесь есть один подводный камень — в некоторых СУБД таблице подзапроса необходимо присваивать псевдоним, а в других нет. Приводимый далее пример использует синтаксис для MySQL, где псевдоним присваивать нужно. Таким псевдонимом является `HEAD_COUNT_TAB` после закрывающей скобки. Присваивание псевдонима также требуется в PostgreSQL и SQL Server, но не требуется в Oracle.

```
select max(HeadCount) as HighestJobHeadCount from
(select job,count(empno) as HeadCount
    from emp
    group by job) head_count_tab
```

Обобщенные табличные выражения

Обобщенные табличные выражения (ОТВ) предназначены для преодоления некоторых ограничений подзапросов и, возможно, лучше всего известны тем, что позволяют исполнению в SQL рекурсивных запросов. Фактически главным поводом для создания ОТВ и было желание получить возможность рекурсии.

Результат следующего запроса такой же, как и рассмотренного ранее подзапроса, — двойная агрегация:

```
with head_count_tab (job,HeadCount) as
(select job,count(empno)
  from emp
  group by job)
select max(HeadCount) as HighestJobHeadCount
  from head_count_tab
```

Хотя этот запрос решает простую задачу, он иллюстрирует основные свойства обобщенных табличных выражений. Посредством оператора WITH вводится производная таблица, при этом в скобках указываются названия столбцов, а сам запрос для получения производной таблицы также заключается в скобки. Можно добавлять дополнительные производные таблицы, разделяя их запятыми и указывая их названия перед соответствующим запросом (процесс, обратный обычному процессу присваивания псевдонимов в SQL).

Поскольку внутренние запросы представляются перед внешним запросом, во многих случаях они также могут быть более удобочитаемыми — они облегчают анализ каждого логического элемента запроса по отдельности, чтобы можно было понять логическую последовательность всего запроса. Конечно же, подобно всему, связанному с созданием кода, это будет зависеть от обстоятельств, и иногда более удобочитаемым будет подзапрос.

С учетом того, что возможность рекурсии является главной причиной создания ОТВ, наилучший способ продемонстрировать такую возможность — это выполнить рекурсивный запрос.

Следующий запрос вычисляет первые 20 значений последовательности Фибоначчи с помощью рекурсивного ОТВ. Обратите внимание на то, что в первой части опорного запроса можно инициализировать значения первой строки виртуальной таблицы:

```
with recursive workingTable ( fibNum, NextNumber, index1)
as
(select 0,1,1
 union all
 select fibNum+nextNumber,fibNum,index1+1
 from anchor
 where index1<20)
select fibNum from workingTable as fib
```

Следующие члены последовательности Фибоначчи вычисляются сложением значений текущего и предыдущего ее членов. Этот результат также можно получить с помощью функции LAG. Но в нашем случае мы создали псевдофункцию LAG, используя два столбца, чтобы учитывать текущее и предыдущее значения. Обратите внимание на ключевое слово RECURSIVE, которое нужно обязательно использовать в MySQL, Oracle и PostgreSQL (но не в SQL Server или DB2). В нашем запросе столбец index1 по большому счету избыточный — в том смысле, что он не используется для вычисления членов последовательности Фибоначчи. Он был включен для того, чтобы было проще задавать номера возвращаемым строкам посредством

оператора `WHERE`. В рекурсивном ОТВ оператор `WHERE` является критически важным, так как без него запрос никогда не завершится. (Хотя в нашем конкретном случае попытка удалить его, скорее всего, вызовет ошибку переполнения, когда значения последовательности превысят размер своего типа данных.)

В простых случаях разница в уровне практичности между подзапросом и ОТВ не очень велика. Оба позволяют вложение запросов или создание более сложных запросов, обращающихся к другим производным таблицам. Но с повышением количества вложенных подзапросов удобочитаемость понижается вследствие сложности понимания значения разных переменных, содержащихся в последовательных уровнях запроса. А поскольку все элементы ОТВ организованы вертикально, значения каждого из них понимать легче.

Подведем итоги

Использование производных таблиц значительно расширяет область действия SQL. И подзапросы, и обобщенные табличные выражения многократно использовались в этой книге, поэтому важно понимать, как они работают, особенно по той причине, что каждая из этих конструкций имеет свой особый синтаксис, которым нужно овладеть, чтобы обеспечить их успешное применение. Рекурсивные ОТВ, которые поддерживаются рассматриваемыми в этой книге СУБД, являются одним из самых больших расширений SQL, предоставляющим множество дополнительных возможностей.

Предметный указатель

D

DB2

- ◇ оператор
 - CASE 300
 - FETCH FIRST 324
 - VALUES 372
 - WITH 300, 334
- ◇ функция
 - COUNT 334
 - CURRENT_DATE 401
 - DATE 329
 - DATENAME 263
 - DAY 257
 - DAYNAME 249, 259, 293, 297
 - DAYOFWEEK 344
 - DAYOFYEAR 285, 296
 - DAYS 247
 - DENSE_RANK 223
 - MAX 279, 281
 - MAX OVER 231
 - MIN OVER 231
 - MOD 326
 - MONTH 255
 - MONTHNAME 344
 - OLAP 565
 - PERCENTILE_CONT 225, 226, 238
 - QUARTER 325
 - REPEAT 431
 - REPLACE 233
 - ROW_NUMBER OVER 321, 324
 - SECOND 289
 - SUBSTR 326
 - SUBSTR 328
 - TRANSLATE 233
 - YEAR 255, 367

M

MySQL

- ◇ оператор: CASE 228
- ◇ расширение: CUBE 453
- ◇ функция
 - ADDDATE 286, 301, 323, 340
 - ADDDATE 332
 - CONCAT 332, 348
 - CUME_DIST 227
 - CURDATE 401
 - CURRENT_DATE 264
 - DATE_ADD 246, 292, 332
 - DATE_FORMAT 260
 - DATEADD 264
 - DATEDIFF 258
 - DATEFORMAT 264
 - DATENAME 265
 - DAY 292, 293, 332
 - DAYNAME 250, 298
 - DAYOFWEEK 306
 - DAYOFYEAR 340
 - DENSE_RANK 223
 - EXTRACT 368
 - GROUPING 453
 - LAST_DAY 284, 292, 293, 332
 - MAX 310
 - MOD 327
 - MONTH 255
 - PERCENTILE_CONT 226, 227
 - STR_TO_DATE 332
 - SUBSTR 327, 331
 - YEAR 255

O

Oracle

- ◇ оператор
 - CONNECT 335
 - CONNECT BY 261, 297, 485, 486, 487, 493
 - ITERATE 372

Oracle (прод.)

- ◇ оператор (прод.)
 - KEEP 224
 - MODEL 371, 372, 507, 508
 - NEXT_DAY 300
 - START WITH 493
 - WITH 293, 297
- ◇ расширение: KEEP 223
- ◇ функции аналитические 565
- ◇ функция
 - ADD_MONTH 515
 - ADD_MONTHS 245, 286, 322, 330, 338
 - CONNECT_BY 489
 - CONNECT_BY_ISLEAF 495, 499
 - CONNECT_BY_ROOT 495, 499, 500
 - COUNT 335, 339, 340
 - DATE_TRUNC 301
 - DECODE 534, 545, 551
 - DENSE_RANK 224
 - INSTR 512, 513
 - ITERATION_NUMBER 372
 - LAST_DAY 279, 282, 291, 292, 305, 330
 - LEAD OVER 273, 350, 351, 361
 - MAX 223, 224, 308
 - MAX OVER 231, 550
 - MEDIAN 226, 239
 - MIN OVER 231
 - MOD 327
 - MODEL 519–521
 - NEXT_DAY 304, 305
 - PERCENTILE_CONT 226
 - RATIO_TO_REPORT 547
 - REPLACE 233
 - ROWNUM 267, 322
 - RPAD 525
 - SUBSTR 327, 329, 512
 - SYS_CONNECT_BY_PATH 485, 487, 489, 491, 501
 - TO_CHAR 250, 267, 282, 289, 294, 297, 460, 515
 - TO_DATE 330
 - TO_NUMBER 460
 - TRANSLATE 233
 - TRUNC 266, 286, 291, 292, 297, 304, 305, 338, 515

P

PostgreSQL

- ◇ ключевое слово: RECURSIVE 298
- ◇ оператор
 - CASE 228
 - CONCAT 484

◇ функция

- CONCAT 488
- DATE_TRUNC 268, 293
- DENSE_RANK 223
- EXTRACT 255, 294, 368
- GENERATE_SERIES 283, 309, 372, 373
- MOD 327
- PERCENTILE_CONT 226, 238
- SUBSTR 327, 330
- TO_CHAR 269, 283, 289, 298
- TO_DATE 331
- TO_NUMBER 289
- TRANSLATE 233

S

SQL 368

SQL Server

- ◇ оператор
 - CASE 302
 - PIVOT 503, 504
 - UNPIVOT 506
 - WITH 302, 323, 337
 - деления по модулю 328
- ◇ функция
 - CAST 285, 333
 - CEILING 428, 458
 - COALESCE 285
 - CONCAT 285
 - COUNT 337
 - DATEADD 246, 286, 288, 292, 299, 323, 333
 - DATEDIFF 256, 258
 - DATENAME 261, 299, 342
 - DATEPART 288, 290, 298, 323
 - DAY 285, 292, 293
 - DAYNAME 295
 - DENSE_RANK 223
 - GET_DATE 285
 - GETDATE 401
 - MAX 311
 - MAX OVER 231
 - MIN OVER 231
 - PERCENTILE_CONT 226, 238
 - REPLACE 233
 - REPLICATE 431
 - SUBSTRING 332
 - TRANSLATE 233
 - YEAR 285

А

Абсолютное отклонение: медианное 240
 Аксиома Фреге 556
 Аналитические функции Oracle 565
 Антиобъединение 80
 Арифметика интервальная 286

В

Ветвления: узел 494
 Вложенные запросы 35, 43
 Внутреннее объединение 70
 Выражение
 ◇ CASE 46, 51, 66, 237, 410, 411, 414, 415,
 419, 422, 426, 427, 433, 437, 440, 454, 456,
 472–474
 ◇ CONNECT BY 294
 ◇ CREATE TABLE 111
 ◇ DELETE 124
 ◇ INSERT 108
 ◇ INSERT ALL 113, 114
 ◇ INSERT FIRST 113, 114
 ◇ MERGE 123
 ◇ ORDER BY 67
 ◇ TRUNCATE 124
 ◇ UPDATE 116
 Выражения обобщенные табличные 30, 581

Г

Группа SQL 554, 555
 Группировка 553

Д

Данные хранилища 477
 Даты: тип TIMESTAMP 246
 Денормализованное представление 418

З

Закон Бенфорда 241–243
 Запросы вложенные 35, 43
 Защищенная таблица по ключу 121

И

Интервальная арифметика 286

К

Кадр 573
 Кадрование: оператор 572, 573, 574
 Ключевое слово
 ◇ DEFAULT 109
 ◇ DISTINCT 95, 392, 393, 556
 ◇ INTERVAL 246
 ◇ OVER 566
 ◇ PRIOR 487
 ◇ RECURSIVE 298, 420, 422, 484
 Команда SHOW INDEX 138
 Концевой узел 494
 Корневой узел 494

М

Медианное абсолютное отклонение 240
 Мода 222

О

Объединение внутреннее 70
 Оконные функции 31, 562, 565, 570, 571
 Оператор
 ◇ ASC 53
 ◇ CASE 228, 300, 302, 303, 395
 ◇ CONCAT 484
 ◇ CONNECT 335
 ◇ CONNECT BY 261, 297, 314, 401, 405, 485,
 486, 487, 493
 ◇ DEFAULT VALUES 109
 ◇ DESC 53
 ◇ DISTINCT 69
 ◇ EXCEPT 74, 75
 ◇ FETCH FIRST 46, 47, 324
 ◇ GROUP BY 209–213, 215, 217, 392, 393,
 431, 438, 439, 472–475, 556, 558, 562–564
 ◇ INTERSECT 73
 ◇ INTO 112
 ◇ IS NULL 50
 ◇ ITERATE 372
 ◇ JOIN 380, 381
 ◇ KEEP 224, 396–398
 ◇ LIKE 52, 111
 ◇ LIMIT 47
 ◇ MINUS 74, 75, 85
 ◇ MODEL 372, 507, 508
 ◇ NEXT_DAY 300
 ◇ NTERSECT 73

Оператор (прод.)

- ◇ ORDER BY 53, 55, 219, 393, 463, 464, 566, 570
 - ◇ OVER 65, 571
 - ◇ PARTITION BY 463, 567–569, 570
 - ◇ PIVOT 503, 504
 - ◇ RANGE BETWEEN 571, 572
 - ◇ START WITH 493
 - ◇ UNION 69, 103
 - ◇ UNION ALL 68, 69, 86, 318, 450, 451, 485, 562
 - ◇ UNPIVOT 506
 - ◇ VALUES 372
 - ◇ WITH 279, 280, 293, 297, 300, 302, 303, 323, 334, 337, 371, 405, 484, 485, 493
 - ◇ деления по модулю 328
 - ◇ кадрирования 572, 573, 574
 - ◇ подстановки
- Операции вычитания множеств 74
- OTB 581
- ◇ рекурсивные 582, 583
- Отклонение медианное абсолютное 240
- Отношение
- ◇ родитель-потомок 485
 - ◇ типа "множество-один" 32

П

Пакет DBMS_RANDOM 49

Парадокс Расселла 556–558

Последовательность Фибоначчи 582

Правило $n-1$ 92

Предикат

- ◇ NOT EXISTS 78, 125
- ◇ NOT IN 125

Представление денормализованное 418

Присваивание псевдонимов 43

Р

Расширение

- ◇ CUBE 442–445, 447, 452, 453
 - ◇ GROUPING SETS 447–449
 - ◇ KEEP 223
 - ◇ NULLS FIRST 63, 65
 - ◇ NULLS LAST 63, 65
 - ◇ ROLLUP 438, 439, 441, 452, 453, 472–476
- Рекурсивные OTB 582, 583

С

Сводные таблицы 33, 146

Слово ключевое

- ◇ DISTINCT 392, 393, 556
 - ◇ OVER 566
 - ◇ PRIOR 487
 - ◇ RECURSIVE 420, 422, 484
- Среднее усеченное 231, 233

Т

Таблица

- ◇ BONUS.TYPE 93
- ◇ DEPT 32
- ◇ EMP 32
- ◇ EMP_BONUS 93
- ◇ защищенная по ключу 121

Таблицы сводные 146

Табличные выражения обобщенные 30, 581

Тип дат TIMESTAMP 246

У

Узел

- ◇ ветвления 494
- ◇ конечной 494
- ◇ корневой 494

Усеченное среднее 231, 233

Ф

Фибоначчи, последовательность 582

Функции

- ◇ ADD_MONTH 515
- ◇ ADD_MONTHS 245, 286, 322, 330, 338
- ◇ ADDDATE 286, 301, 323, 332, 340
- ◇ AVG 210
- ◇ CAST 188, 285, 333
- ◇ CEIL 428, 458
- ◇ CEILING 428, 458
- ◇ CHARINDEX 178
- ◇ COALESCE 50, 51, 105, 210, 230, 231, 285, 359, 438, 441
- ◇ CONCAT 285, 332, 348, 488
- ◇ CONCAT_WS 164
- ◇ CONNECT_BY 489
- ◇ CONNECT_BY_ISLEAF 495, 499
- ◇ CONNECT_BY_ROOT 495, 499, 500

- ◇ COUNT 215–217, 334, 335, 337, 339, 340, 432, 554, 558, 560
- ◇ COUNT OVER 277, 462, 463, 547
- ◇ CUME_DIST 227
- ◇ CURDATE 401
- ◇ CURRENT_DATE 264, 307, 401
- ◇ CURRENT_DAY 306
- ◇ DATE 329
- ◇ DATE_ADD 246, 292, 332
- ◇ DATE_FORMAT 260, 344
- ◇ DATE_TRUNC 268, 293, 301
- ◇ DATEADD 246, 264, 286, 288, 292, 299, 323, 333
- ◇ DATEDIFF 248, 256, 257, 258
- ◇ DATEFORMAT 264
- ◇ DATENAME 261, 299, 263, 265, 342
- ◇ DATEPART 288, 290, 298, 323
- ◇ DAY 257, 285, 292, 293, 332
- ◇ DAYNAME 249, 250, 293, 295, 297, 298
- ◇ DAYOFWEEK 306, 344
- ◇ DAYOFYEAR 285, 288, 296, 340
- ◇ DAYS 247, 272
- ◇ DB2.SUM OVER 228
- ◇ DB2MONTH 289
- ◇ DECODE 424, 534, 545, 551
- ◇ DENSE_RANK 223, 224, 383, 384, 398
- ◇ DENSE_RANK OVER 391, 396, 522, 524
- ◇ DIFFERENCE 205
- ◇ EXCEPT 85, 88
- ◇ EXTRACT 255, 294, 368
- ◇ GENERATE_SERIES 283, 309, 372, 373, 402, 407
- ◇ GET_DATE 285
- ◇ GETDATE 401
- ◇ GROUP_CONCAT 172–174, 183, 185, 193
- ◇ GROUPING 439, 440, 447, 453, 472, 475
- ◇ INSSTR 189
- ◇ INSTR 177, 200, 202, 512, 513, 518, 544
- ◇ ITERATION_NUMBER 372
- ◇ LAG 221
- ◇ LAG OVER 356, 364, 365, 388, 390, 423, 424
- ◇ LASE 396
- ◇ LAST_DAY 279, 282, 284, 291, 292, 293, 305, 330, 332
- ◇ LEAD OVER 273, 350, 351, 354, 355, 358, 359, 361, 362, 363, 386, 387, 388, 390
- ◇ LEN 165
- ◇ LENGTN 165
- ◇ LIST_AGG 173, 174
- ◇ LN 220
- ◇ LOCATE 176
- ◇ LOG 220
- ◇ LPAD 431
- ◇ MAX 211, 212, 223, 224, 252, 279, 281, 308, 311, 320, 364, 366, 412, 414, 415, 433, 434
- ◇ MAX OVER 231, 384, 385, 389, 395–398, 436, 437, 550
- ◇ MEDIAN 226, 239
- ◇ MIN 211, 212, 364, 366
- ◇ MIN OVER 231, 369, 384, 385, 389, 436, 437
- ◇ MINUS 85, 88
- ◇ MOD 326, 327, 378
- ◇ MODEL 519–521
- ◇ MONTH 255, 303
- ◇ MONTHNAME 344
- ◇ MONTHS_BETWEEN 255, 257
- ◇ MySQLMAX 310
- ◇ NEWID 49
- ◇ NEXT_DAY 304, 305
- ◇ NEXTDAY 306
- ◇ NTILE 430
- ◇ OLAP 565
- ◇ OVER 383
- ◇ PARTITION BY 392
- ◇ PDATE_TRUNC 287
- ◇ PERCENTILE_CONT 225–227, 238
- ◇ POSSTR 188
- ◇ QUARTER 325
- ◇ RAND 48, 49
- ◇ RANDOM 49
- ◇ RATIO_TO_REPORT 547, 548
- ◇ REGEXP_LIKE 208
- ◇ REGEXP_REPLACE 208
- ◇ REPEAT 431
- ◇ REPLACE 58, 144, 150–154, 161, 162, 166, 167, 169, 170, 188, 190–192, 233, 234
- ◇ REPLICATE 431
- ◇ ROW_NUMBER 175, 176, 178, 184, 198, 277, 383, 392, 393
- ◇ ROW_NUMBER OVER 321, 324, 375, 376, 378, 392, 413, 414, 416, 420, 421, 428, 433, 457, 523
- ◇ ROWNUM 47, 49, 177, 267, 322, 377
- ◇ RPAD 525
- ◇ RTRIM 342
- ◇ SECOND 289
- ◇ SIGN 496, 497, 499
- ◇ SOUNDEX 205

Функции (прод.)

- ◇ SPLIT_PART 177, 181, 196, 202
- ◇ STR_TO_DATE 332
- ◇ STRING_ADD 172
- ◇ STRING_AGG 173, 174, 183, 184, 196
- ◇ STRING_SPLIT 196, 199
- ◇ STRPOS 190
- ◇ SUBSTR 56, 146, 165, 176, 177, 198, 200, 202, 203, 326–331, 512
- ◇ SUBSTR_INDEX 202
- ◇ SUBSTRING 56, 165, 178, 332
- ◇ SUBSTRING_INDEX 163, 179, 199
- ◇ SUM 93, 214, 410, 426, 427, 440, 449, 468, 470
- ◇ SUM OVER 95, 97, 99, 218, 219, 236, 464
- ◇ SYS_CONNECT_BY_PATH 173, 175, 183, 186, 485, 487, 489, 491, 501
- ◇ TO_CHAR 250, 267, 269, 282, 283, 289, 294, 297, 342, 344, 460, 515

- ◇ TO_DATE 330, 331
- ◇ TO_NUMBER 289, 424, 460
- ◇ TRANSLATE 58, 59, 150–155, 166–169, 188–192, 233–235, 516, 517
- ◇ TRUNC 266, 286, 291, 292, 297, 304, 305, 338, 515
- ◇ VALUE 49
- ◇ YEAR 255, 285, 367
- ◇ оконные 31

X

Хранилища данных 477

Э

Эквидобъединения 70

Об авторах

Энтони Молинаро (Anthony Molinaro) работает специалистом по данным в компании Johnson & Johnson. В настоящее время он занимает должность руководителя группы анализа эмпирических данных по здравоохранению (Observational Health Data Analytics) в отделе исследований и разработок Janssen. Основной областью его исследований являются непараметрические методы, анализ временных рядов и определение характеристик крупномасштабных баз данных и их преобразование. Он состоит членом открытого научного общества OHDSI (Observational Health Data Sciences and Informatics), занимающегося научным и информационным обеспечением данных по здравоохранению. Энтони получил диплом бакалавра гуманитарных наук по математике и диплом магистра гуманитарных наук по прикладной математике и статистике в колледже Hunter городского университета Нью-Йорка. Он проживает в Нью-Йорке с женой Джорджией и двумя дочерьми, Кики и Конни.

Роберт де Грааф (Robert de Graaf) получил диплом инженера и по завершении обучения работал в обрабатывающей промышленности. Работая инженером, Роберт открыл для себя мощь статистики для решения практических задач и продолжил высшее образование, получив диплом магистра по статистике, как раз вовремя, чтобы воспользоваться им в наступившем буме интеллектуальной обработки данных. С 2013 года он работает в качестве главного специалиста по обработке данных в компании RightShip. Роберт является автором книги «Managing Your Data Science Projects» («Управление проектами интеллектуальной обработки данных»), вышедшей в издательстве Apress.

Об обложке

На обложке этой книги изображена ящерица агама (Agamid lizard). Эти ящерицы принадлежат к семейству агамовых (Agamidae), и их встречается в природе более 300 видов. Ареал обитания агамовых — Африка, Азия, Австралия и Южная Европа. Их часто можно встретить в прибрежных и скалистых горных регионах с сухим или полусухим климатом — от жарких пустынь до теплых влажных тропических лесов.

Агамы характеризуются сильными ногами и — некоторые их разновидности — способностью менять цвет. В отличие от других видов ящериц, агамиды не могут регенерировать свои хвосты, если они их потеряют. Агамы обычно привязаны к своей территории и предпочитают жить группами, хотя перенаселенность может привести к стрессу и потере аппетита, особенно когда животным негде спрятаться. Из-за перенаселенности иногда между самцами агам возникают боевые столкновения, в результате которых они нередко получают травмы, связанные с утратой пальцев ног и хвоста.

Некоторые виды агамид популярны в качестве домашних животных. Среди них Бородатый дракон (Pogona). Спокойные, но любопытные, эти существа вырастают примерно до 20 дюймов. Даже при их небольшом росте они все еще считаются «гигантскими» ящерицами и поэтому требуют для своего содержания достаточно места. Голова бородатой ящерицы имеет треугольную форму со множеством выступающих от подбородка шипов. Эти шипы напоминают усы и бороду — отсюда и название. Бородатые драконы открывают пасть и выставляют свои колючие бороды, чтобы напугать хищников и других бородатых. Они также могут расплющивать свое тело, чтобы казаться больше. Став домашними животными, они перестают показывать «бороду», если им комфортно жить со своими хозяевами, и место для них удобно оборудовано. Хотя они в основном родом из Австралии, бородатые драконы, продаваемые дилерами в США, являются потомками животных, завезенных из Европы. Это связано со строгими законами Австралии об экспорте диких животных.

Еще один пример агамидной ящерицы — летающая ящерица (*Draco volans*). Ее длинное — чуть менее 12 дюймов — тонкое тело снабжено лоскутами кожи, натянутыми вдоль ребер. Самец летающей ящерицы претендует на свою территорию от двух до трех деревьев, на каждом из которых живут от одной до трех самок. Чтобы перемещаться из одного места в другое, он планирует с деревьев или других высоких мест, расправляя свои кожные лоскуты, как крылья. Обычно они не летают ни в дождь, ни в ветер. В случае угрозы летающая ящерица может также растянуть свои кожаные «крылья», чтобы казаться больше.

Очень интересная разновидность семейства агамовых — это агама с красной головкой (*Agama agama*), обитающая в Африке к югу от Сахары. Эти существа часто живут группами от 10 до 20 особей, причем старший самец выступает в роли лидера группы. Ночью их окраска темно-коричневая, но на рассвете она меняется на светло-голубую, оставляя ярко-оранжевыми голову и хвост. Цвет их кожи также меняется в зависимости от настроения — например, когда самцы дерутся, их головы становятся коричневыми, а вдоль тела появляются белые пятна.

Многие представители животного мира, изображенные на обложках книг издательства O'Reilly, находятся под угрозой исчезновения. Все они чрезвычайно важны для нашей планеты.

Иллюстрацию на обложке создала Карен Монтгомери (Karen Montgomery) на основе черно-белой гравюры (исходный источник изображения неизвестен).

SQL. Сборник рецептов

Возможно, вы знаете основы SQL, но умеете ли вы пользоваться всей выразительной мощностью этого языка? Во втором издании книги применяется очень практичный подход к языку SQL, позволяющий создавать большие хранилища данных и управлять ими. Обновленная версия книги основана на практических примерах для разных версий SQL, включая Oracle, DB2, SQL Server, MySQL и PostgreSQL.

Этот сборник рецептов послужит ценным руководством по решению повседневных задач для широкого круга пользователей — программистов приложений SQL, специалистов по работе с данными, администраторов баз данных и даже тех, кто работает с SQL нерегулярно.

Второе издание включает:

- Полностью обновленные рецепты, учитывающие более широкое использование оконных функций в SQL-приложениях
- Дополнительные примеры, показывающие обширное использование обобщенных табличных выражений (ОТВ) для создания более удобочитаемых и простых решений
- Новые решения, которые делают SQL более полезным для пользователей, не являющихся экспертами в области СУБД, включая специалистов по работе с данными
- Расширенные выражения для обработки чисел и строк

«Здорово видеть обновленный сборник рецептов SQL, дополненный современными SQL-темами, включая оконные функции, обобщенные табличные выражения и иерархические запросы».

— Томас Нильд (Thomas Nield)
Автор книги *Getting Started with SQL*,
издательство O'Reilly

«Работа Энтони и Роберта дарит читателю эмоции, которые раньше не вызывала ни одна книга по SQL. Эффективные и действенные решения подаются размеренным темпом, с последующими советами, закрепляющими и дополняющими предшествующий материал».

— Скотт Хейнс (Scott Haines)
Ведущий инженер-программист,
компания Twilio

Энтони Молинаро (Anthony Molinaro) — специалист по данным в компании Johnson & Johnson. Занимается исследованиями непараметрических рядов, анализом временных рядов и характеристик крупномасштабных баз данных, а также их преобразованием.

Роберт де Грааф (Robert de Graaf) — занимает должность главного специалиста по данным в компании RightShip.

ISBN 978-5-9775-6759-6



191036, Санкт-Петербург,
Гончарная ул., 20
Тел.: (812) 717-10-50,
339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru