

Изучаем **SQL** и **MySQL**

С легкостью извлекайте данные и манипулируйте ими с помощью команд SQL

Ашвин Паджанкар



Изучаем SQL и MySQL

*С легкостью извлекайте данные и
манипулируйте ими с помощью
команд SQL*

Ашвин Паджанкар



ПЕРВОЕ ИЗДАНИЕ 2020 ГОД

Copyright © BPB Publications, Индия

ISBN: 978-93-89898-088

Все права защищены. Никакая часть этой публикации не может быть воспроизведена или распространена в любой форме и любыми средствами, а также сохранена в базе данных или поисковой системе без предварительного письменного разрешения издателя, за исключением списков программ, которые могут быть введены, сохранены и выполнены в компьютерной системе, но они не могут быть воспроизведены средствами публикации.

ОГРАНИЧЕНИЯ ОТВЕТСТВЕННОСТИ И ОТКАЗ ОТ ГАРАНТИИ

Информация, содержащаяся в этой книге, соответствует действительности и соответствует знаниям автора и издателя. Автор приложил все усилия для обеспечения точности этих публикаций, но не может нести ответственность за любые потери или ущерб, возникшие в результате использования любой информации в этой книге.

Все торговые марки, упомянутые в книге, признаны собственностью соответствующих владельцев.

Дистрибьюторы:

BPB PUBLICATIONS

20, Ansari Road, Darya Ganj

New Delhi-110002

Ph: 23254990/23254991

MICRO MEDIA

Shop No. 5, Mahendra Chambers,

150 DN Rd. Next to Capital Cinema,

V.T. (C.S.T.) Station, MUMBAI-400 001

Ph: 22078296/22078297

DECCAN AGENCIES

4-3-329, Bank Street, Hyderabad-

500195

Ph: 24756967/24756400

BPB BOOK CENTRE

376 Old Lajpat Rai Market, Delhi-

110006

Ph: 23861747

Опубликовано Манишем Джайном для BPB Publications, 20 Ansari Road, Darya Ganj, New Delhi-110002 и напечатано им в Repro India Ltd, Мумбаи

Посвящается

Джаянту Нарликару

Выдающемуся индийскому астрофизику

Об авторе

Эшвин — опытный ветеран, который последние 25 лет работает в сфере STEM (наука, технологии, инженерия и математика). За свою карьеру Эшвин более 7 лет проработал сотрудником в различных ИТ-компаниях и компаниях, занимающихся разработкой программного обеспечения. Он написал более 2 дюжин книг по Arduino, программированию на Python, компьютерному зрению, Интернету вещей, базам данных и другим популярным темам для ВРВ и других международных изданий. Он рецензировал множество других технических книг. Также он создает курсы для ВРВ и других платформ и обучает онлайн около 60 000 студентов.

С 2017 года Эшвин работает фрилансером. Впервые писательский вкус он почувствовал в 2015 году, когда написал свою первую книгу о Raspberry Pi. В свободное время Эшвин снимает видео для своего Youtube-канала, у которого сейчас более 10 000 подписчиков.

В свободное от работы время Эшвин выполняет роль посла STEM, помогая, тренируя и наставляя молодых людей, желающих сделать карьеру в сфере технологий.

О рецензенте

Доктор Абдул Азиз, в настоящее время работающий штатным инженером в VMware Software India Pvt Ltd, окончил ИИТ-Хайдарабад со степенью CSE (с отличием в области инженерии данных) в 2007 году. Азиз увлечен проектами с открытым исходным кодом, а в свободное время работает над проектами ядра Linux, открытого стека и облачных вычислений.

Азиз читал лекции и выступал на темы IPv6, сетевого стека Linux, облачных технологий в таких престижных учреждениях, как NIT-Warangal, PESIT, RVCE. Азиз получил патент на многоузловую систему виртуальной коммутации (USPTO DN/20140204805). В прошлом Азиз занимал должность инженера по разработке программного обеспечения и ведущего специалиста в Cisco и Microsoft в области виртуализации, технологий центров обработки данных, многоуровневой коммутации и интернет-технологий.

Слова благодарности

Есть несколько человек, которых я хочу поблагодарить за постоянную поддержку, которую они мне оказывали во время написания этой и других книг с помощью ВРВ Publications.

Эта книга не появилась бы на свет, если бы я не получил поддержку со стороны некоторых ключевых людей из ВРВ Publications. Я выражаю благодарность команде ВРВ. Без их поддержки я бы никогда не завершил эту книгу. Это моя четвертая книга с ВРВ, и я планирую написать еще.

Наконец, я хотел бы поблагодарить технического рецензента за его ценный вклад.

Предисловие

Я профессионально работаю с различными СУБД с тех пор, как начал свою официальную профессиональную карьеру в июне 2007 года. Я много работал с различными ОРСУБД (системами управления объектно-реляционными базами данных), такими как Oracle, IBM DB2, Teradata, MySQL и MariaDB. А с 2010 года я работаю и с MySQL.

Мы используем реляционные базы данных и ОРСУБД повсюду, от корпоративных приложений до мобильных приложений. Они являются неотъемлемой частью жизни и средств к существованию программистов. Хотя не все тратят большую часть своего рабочего времени непосредственно на работу с базами данных, программисты регулярно обращаются к сервисам баз данных для своих приложений. Большинство организаций используют РСУБД для структурированного хранения данных.

SQL (структурированный язык запросов) — это лингва-франка в мире РСУБД и ОРСУБД. Почти все основное программное обеспечение ОРСУБД является разновидностью SQL, реализованное на его ядре, так что администраторы и обычные пользователи могут легко взаимодействовать с ОРСУБД и данными. SQL использовался практически во всех операционных системах и вычислительных платформах, включая серверы, настольные компьютеры, карманные компьютеры и мобильные устройства. Программное обеспечение ОРСУБД доступно для всех типов вычислительных устройств, а SQL является общей основой для них.

Из-за множества разновидностей SQL, предоставляемых различным программным обеспечением РСУБД, начинающие разработчики часто путаются с выбором РСУБД для изучения SQL. В то время как поставщики программного обеспечения, такие как Oracle, предоставили бесплатную (для использования в некоммерческих целях), но некорпоративную версию своего проприетарного программного обеспечения, MySQL и ее ответвление MariaDB могут свободно использоваться для проектов даже в коммерческих целях. Мы можем использовать все их возможности на корпоративном уровне и в режиме реального времени. Новичкам часто советуют начинать изучение SQL с

MySQL или MariaDB. Мы можем изучить SQL с новейшим синтаксисом ANSI с помощью баз данных MySQL и MariaDB. Эти базы данных доступны в Windows и Linux. В этой книге мы узнаем, как установить MySQL и MariaDB на платформе Windows. Мы также узнаем, как установить MariaDB на Raspberry Pi Raspbian OS (недавно переименованную в Raspberry Pi OS). Мы изучим различные интерфейсы, такие как запросы MySQL и MySQL Workbench, для подключения к серверам MySQL и MariaDB.

Эта книга чрезвычайно полезна для новичков, поскольку она начинается с нуля и с самого начала охватывает самую основную часть. Если у вас есть достаточный технический опыт и вам не хватает детальных знаний SQL, эта книга — правильный выбор для начала работы с SQL. Эта книга очень полезна для разработчиков, инженеров, специалистов по обработке данных и администраторов баз данных начального уровня.

Глава 1 знакомит с проектами MySQL и MariaDB, а также с концепциями баз данных.

Глава 2 описывает инструменты для подключения к MySQL и MariaDB. Здесь также объясняется, как загрузить и установить примеры схем на серверах MySQL и MariaDB для нашей практики.

Глава 3 подробно обсуждает запрос SELECT. В ней также объясняется концепция строк NULL и DISTINCT.

Глава 4 — это ключевая глава, в которой подробно обсуждается, чего можно достичь благодаря оператору WHERE. Она ориентирован на различные операторы. Подробно объясняется операция выбора.

Глава 5 также является ключевой главой, в которой подробно обсуждается, что такое функция одной строки. В ней обсуждается обработка NULL и вызовы вложенных функций для однострочной функции.

Глава 6 описывает некоторые более сложные функции, известные как групповые функции. Она исследует операторы Group by и Having в SQL.

Глава 7 знакомит читателей с техникой объединения данных из различных источников. Этот метод известен как соединение или Join. В главе рассматривается синтаксис ANSI и более старый синтаксис для различных типов соединений.

Глава 8 описывает, как написать запрос внутри запроса. Подробно исследуются два типа подзапросов.

Глава 9 описывает, как выполнять DDL и DML. Она также подробно исследует концепцию транзакций.

Глава 10 описывает, как создавать представления. В ней подробно рассматриваются оба типа представлений.

Глава 11 описывает, как соединить Python 3 с MySQL и MariaDB. В ней рассматривается, как загрузить фрейм данных pandas данными из таблицы MySQL.

Загрузка пакета кода и цветных изображений:

Пожалуйста, перейдите по ссылке, чтобы скачать
пакет кодов и цветные изображения книги:

<https://rebrand.ly/pywh611>

Ошибки

Мы очень гордимся своей работой в BPB Publications и следуем лучшим практикам, чтобы обеспечить точность нашего контента и предоставить нашим подписчикам приятные впечатления от чтения. Наши читатели — это наше зеркало, и мы используем их вклад, чтобы локализовать и исправить человеческие ошибки, если таковые имели место в процессе публикации. Чтобы мы могли поддерживать качество и помочь нам связаться с читателями, у которых могут возникнуть трудности из-за непредвиденных ошибок, напишите нам по адресу :

errata@bpbonline.com

Семья BPB Publications высоко ценит вашу поддержку, предложения и ОТЗЫВЫ.

Оглавление

https://t.me/it_boooks/2

1. Введение и установка

[Структура](#)

[Цель](#)

[Базы данных, СУБД и SQL](#)

[Немного практики с SQL](#)

[MySQL](#)

[Установка MySQL в среде Windows](#)

[MariaDB](#)

[Установка в среде Windows](#)

[Установка в среде Debian и Raspberry Pi Raspbian Linux](#)

[Заключение](#)

[Что следует помнить](#)

[Вопросы](#)

[Ответы к вопросам](#)

[Вопросы](#)

[Ключевые термины](#)

2. Начало работы с MySQL

[Структура](#)

[Цель](#)

[Подключение к экземплярам серверов MySQL и MariaDB](#)

[Подключение с помощью MySQL Workbench](#)

[Подключени с помощью Heidi SQL](#)

[Подключение с помощью командной строки](#)

[Подключение к удаленному экземпляру](#)

[Несколько основных запросов](#)

[Упражнение](#)

[Заключение](#)

[Что следует помнить](#)

[Вопросы](#)

[Ответы на вопросы](#)

[Вопросы](#)

[Ключевые термины](#)

[3.Начало работы с SQL-запросами](#)

[Структура](#)

[Цель](#)

[Начало работы с простыми операторами SQL](#)

[Выбор и проецирование](#)

[Арифметические операторы и их приоритет](#)

[Псевдонимы столбцов](#)

[NULL](#)

[DISTINCT](#)

[Заключение](#)

[Что следует помнить](#)

[Вопросы](#)

[Ответы на вопросы](#)

[Вопросы](#)

[Ключевые термины](#)

[4.The WHERE Clause in Detail](#)

[Структура](#)

[Цель](#)

[Оператор WHERE](#)

[Операторы сравнения](#)

[Оператор LIKE](#)

[Сравнение с NULL](#)

[Логические операции](#)

[Сортировка набора результатов](#)

[Работа с датами](#)

[Заключение](#)

[Что следует помнить](#)

[Вопросы](#)

[Ответы на вопросы](#)

[Вопросы](#)

[Ключевые термины](#)

[5.Однострочные функции](#)

[Структура](#)

[Цель](#)

[Однорочные функции](#)

[Двойная таблица](#)

[Функции управления регистром](#)

[Функции управления символами](#)

[Функции манипулирования числами](#)

[Функции манипулирования датами](#)

[Дата и время в строку и наоборот](#)

[Обработка NULL](#)

[Оператор CASE](#)

[Вложенные однорочные функции](#)

[Заключение](#)

[Что следует помнить](#)

[Вопросы](#)

[Ответы на вопросы](#)

[Вопросы](#)

[Ключевые термины](#)

6. Групповые функции

[Структура](#)

[Цель](#)

[Групповые функции](#)

[Оператор GROUP BY](#)

[HAVING](#)

[Заключение](#)

[Что следует помнить](#)

[Вопросы](#)

[Ответы на вопросы](#)

[Вопросы](#)

[Ключевые термины](#)

7. Объединения в MySQL

[Структура](#)

[Цель](#)

[Концепция объединения](#)

[Перекрестное объединение](#)

[Простые/внутренние и естественные объединения](#)

[Внешние объединения](#)

[Объединение с несколькими условиями и несколькими источниками](#)

[Самообъединение](#)

[Неэквивалентные объединения](#)

[Заключение](#)

[Что следует помнить](#)

[Вопросы](#)

[Ответы на вопросы](#)

[Вопросы](#)

[Ключевые термины](#)

8. Подзапросы

[Структура](#)

[Цель](#)

[Подзапрос](#)

[Связанный подзапрос](#)

[Заключение](#)

[Что следует помнить](#)

[Вопросы](#)

[Ответы на вопросы](#)

[Вопросы](#)

[Ключевые термины](#)

9. DDL, DML и транзакции

[Структура](#)

[Цель](#)

[Таблицы и словарь данных](#)

[Операторы DDL](#)

[Транзакции и операторы DML](#)

[Операция усечения](#)

[Создание таблицы](#)

[Ограничения](#)

[Удаление базы данных](#)

[Заключение](#)

[Что следует помнить](#)

[Вопросы](#)

[Ответы на вопросы](#)

[Вопросы](#)
[Ключевые](#)
[термины](#)

10. Представления

[Структура](#)
[Цель](#)
[Концепция представлений](#)
[Простые представления](#)
[Сложные представления](#)
[Заключение](#)
[Что следует помнить](#)
[Вопросы](#)
[Ответы на вопросы](#)
[Вопросы](#)
[Ключевые термины](#)

11. Python 3, MySQL и Pandas

[Структура](#)
[Цель](#)
[Установка Python 3](#)
[Запуск программы Python 3](#)
[MySQL и Python 3](#)
[MySQL и Pandas](#)
[Заключение](#)
[Что следует помнить](#)
[Вопросы](#)
[Ответы на вопросы](#)
[Вопросы](#)
[Ключевые термины](#)

ГЛАВА 1

Введение и установка

https://t.me/it_boooks/2

Рекомендую всем читателям прочитать предисловие и оглавление. В книге содержится много информации о том, что нас ждет. Так что, если вы не читали, рекомендую ознакомиться.

В этой главе мы собираемся начать увлекательное путешествие по изучению SQL с широко используемым программным обеспечением баз данных с открытым исходным кодом MySQL и MariaDB. Прежде чем мы приступим к их практическому использованию, мы также рассмотрим несколько основных концепций, которые весьма важны. Наконец, мы подробно узнаем, как установить MySQL и MariaDB на основные платформы ОС. Итак, давайте начнем невероятное путешествие по изучению SQL с MySQL и MariaDB.

Структура

В этой главе мы изучим следующие темы:

- Основные понятия, связанные с базами данных, СУБД и SQL
- Практический опыт работы с SQL с помощью «Tryit Online SQL Editor»
- MySQL и ее установка в среде Windows
- MariaDB и ее установка в среде Windows и Linux

Цель

Цель этой главы — познакомить читателей с основными понятиями, связанными с базами данных. Кроме того, читатели смогут установить MySQL и MariaDB на разные операционные системы. Читатели также научатся использовать онлайн-редактором SQL — Tryit Online SQL Editor. Все программное обеспечение, которое мы научимся устанавливать в этой главе, будет использоваться на протяжении всей оставшейся книги.

Базы данных, СУБД и SQL

База данных — это данные, собранные и хранящиеся в организованной форме. Их можно хранить, получать к ним доступ и обрабатывать как в электронном, так и в бумажном виде. До появления компьютеров люди и организации хранили записи в табличной форме в книгах и документах. Это были предшественники современных систем реляционных баз данных. Сегодня, в 21 веке, мы храним и обрабатываем практически все базы данных в электронном виде.

Базы данных, использующие реляционную модель для хранения и обработки данных, называются реляционными базами данных. Э. Ф. Кодд в 1970 году впервые предложил реляционную модель данных в своей исследовательской работе «**Реляционная модель данных для больших общих банков данных**». Почти все реляционные базы данных используют для обработки данных, хранящихся в базе данных, **структурированный язык запросов (SQL)**. В реляционной модели связанные сущности хранятся в табличной структуре данных, известной как таблица. Современные реляционные базы данных часто являются объектно-реляционными базами данных. Они могут легко взаимодействовать с языками программирования, поддерживающими концепцию объектов. Существуют также другие модели исторических и современных данных. Ниже приведен список некоторых из них:

- ♦ Иерархическая модель базы данных
- ♦ Сетевая модель
- ♦ Объектная модель
- ♦ Документальная модель
- ♦ Модель «ключ-значение»
- ♦ Ассоциативная модель
- ♦ Корреляционная модель
- ♦ Многомерная модель
- ♦ Многозначная модель
- ♦ Семантическая модель
- ♦ XML-база данных
- ♦ Именованный граф (GraphDB)
- ♦ Triplestore/инфраструктура распределения ресурсов (RDF)

Система управления базой данных (СУБД) — это программное обеспечение, которое взаимодействует с базой данных, операционной системой, языками программирования и конечными пользователями для сбора, хранения, обработки и анализа данных, хранящихся в базе данных. СУБД, работающая с реляционными базами данных, — это реляционная СУБД, и аналогично СУБД, работающая с объектно-реляционной моделью данных, — это объектно-реляционная СУБД.

СУБД и базы данных — это обширные темы, которые сами по себе требуют изучения нескольких специальных книг. При изучении SQL мы встретимся со многими концепциями СУБД. Мы будем усваивать эти концепции везде в этой книге, где мы с ними встретимся.

Немного практики с SQL

В предыдущем разделе мы узнали, что SQL — это язык запросов для взаимодействия с базой данных. Мы можем попробовать провести демонстрацию SQL, даже не устанавливая какое-либо программное обеспечение СУБД. Все, что нам нужно, это компьютер с подключением к сети Интернет. На сайте [w3schools.com](https://www.w3schools.com) есть очень хороший онлайн-редактор SQL и образец базы данных с несколькими таблицами. Он известен как Tryit Online SQL Editor и предназначен для быстрой отработки навыков SQL. Хотя это не полнофункциональный продукт для работы с базами данных, мы, безусловно, можем здесь попрактиковаться в выполнении множества запросов. Итак, давайте начнем. Откройте веб-браузер на своем компьютере и перейдите по URL-адресу https://www.w3schools.com/sql/trysql.asp?filename=trysql_op_in. В результате откроется следующая страница:



The screenshot shows the W3Schools SQL Editor interface. On the left, there is a text area for entering an SQL statement. The statement entered is: `SELECT * FROM Customers WHERE (City IN ('Paris','London'))`. Below the text area is a green button labeled "Run SQL". Below the button is a "Result" section with a text area for the output. The output text reads: "W3Schools has created an SQL database in your browser. The menu to the right displays the database, and will reflect any changes. Feel free to experiment with any SQL statement. You can restore the database at any time." On the right side of the interface, there is a section titled "Your Database:" which contains a table listing tables and their record counts.

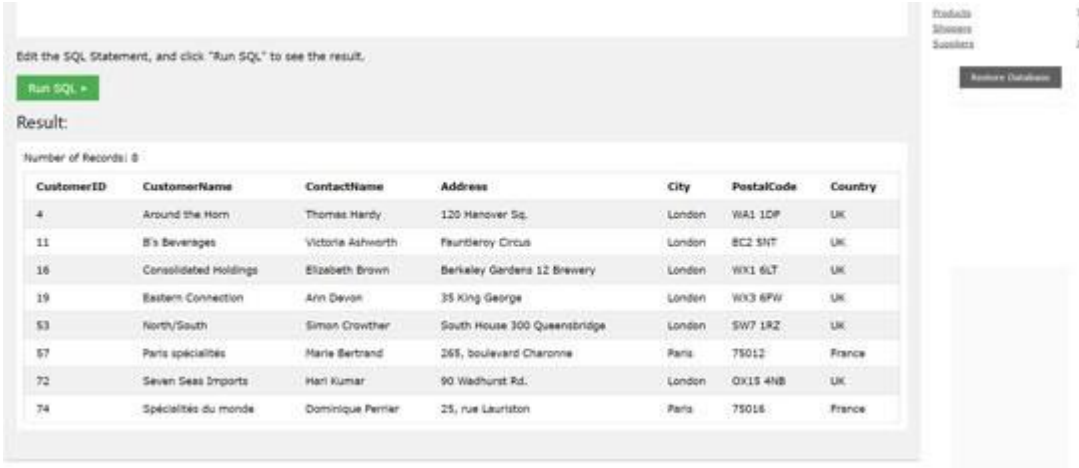
Tablename	Records
Customers	91
Categories	8
Employees	10
OrderDetails	118
Orders	196
Products	77
Suppliers	3
Shippers	29

Рис. 1.1

Мы видим текстовую область, занимающую большую часть страницы. У нас также есть текст `SELECT * FROM Customers WHERE City IN ('Paris', 'London')`; уже присутствует в текстовой области. Этот текст известен как запрос. Запрос — это оператор SQL, который запускается ядром базы данных для выполнения некоторой операции с базой данных. В данном случае база данных является локальной и находится в нашем браузере и оперативной памяти (ОЗУ). Мы видим зеленую кнопку с надписью `Run SQL`.

Справа отображаются таблицы с членами базы данных и количество записей в этих таблицах, перечисленных напротив них.

Давайте запустим запрос в текстовой области. Он даст следующий результат:



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
11	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London	EC2 3NT	UK
16	Consolidated Holdings	Elizabeth Brown	Berkeley Gardens 12 Brewery	London	WX1 6LT	UK
19	Eastern Connection	Ann Devon	35 King George	London	WX3 6FW	UK
53	North/South	Simon Crowther	South House 300 Queensbridge	London	SW7 1RZ	UK
57	Paris spécialités	Marie Bertrand	265, boulevard Charonne	Paris	75012	France
72	Seven Seas Imports	Hari Kumar	90 Wadhurst Rd.	London	OX15 4NB	UK
74	Spécialités du monde	Dominique Perrier	25, rue Lauriston	Paris	75016	France

Рис. 1.2

Запрос простой. Он выбирает все записи из таблицы `Customers`, где в столбце `City` указаны значения `London` и `Paris`. Измените запрос следующим образом `SELECT * FROM Customers;` и выполните его еще раз. Он покажет все записи в таблице `Customers`. Точка с запятой, в конце, не обязательна. Удалите точку с запятой и запустите запрос.

Запросы не чувствительны к регистру, за исключением случаев, когда они ссылаются на данные в таблицах. Мы можем написать приведенный выше запрос, например, так `select * from customers`, он будет выполняться без каких-либо проблем. Однако строковые данные в базе данных чувствительны к регистру. Запрос `select * from customers where city in ('paris', 'london')` — это не то же самое, что `select * from customers where city in ('Paris', 'London')`. Запустите оба запроса по отдельности и обратите внимание, что более

ранний (тот, в котором названия городов написаны строчными буквами) не дает ни одной строки на выходе, как в столбце `City`, таблицы `Customer`, все названия городов имеют первую букву названия столицы. Как мы узнали ранее, данные, хранящиеся в таблицах базы данных, всегда чувствительны к регистру, тогда как ключевые слова и наименование таблицы в запросе нечувствительны к регистру. С этим мы можем попрактиковаться в большем количестве запросов. Но нам нужны настоящие продукты баз данных, чтобы изучить все возможности SQL.

MySQL

MySQL — бесплатная реляционная СУБД с открытым исходным кодом. Его создала шведская компания **MySQL AB**. MySQL AB была основана Дэвидом Аксмарком, Алланом Ларссоном и Майклом «Монти» Видениусом. *My* - имя дочери Майкла. Именно поэтому продукт получил название MySQL. MySQL является важным компонентом стека веб-разработки **Linux, Apache, MySQL, Perl/Python/PHP (LAMP)**. Его используют многие правительственные учреждения, включая NASA (ссылка: <https://www.mysql.com/fr/customers/industry/?id=69>) и крупные организации, такие как Facebook, Twitter и Youtube. Подробную информацию о MySQL можно найти на <https://www.mysql.com/>. Разработка MySQL началась в 1994 году. На следующей временной шкале показаны основные вехи развития MySQL:

- Первый внутренний выпуск *23 мая 1995 г.*
- Версия для Windows была выпущена *8 января 1998 г.* для Windows 95 и NT
- **Версия 3.21:** рабочий релиз *1998 года*
- **Версия 4.0:** бета-релиз от *августа 2002 г.*, рабочий релиз - *март 2003 г.*
- **Версия 5.0:** бета-релиз от *марта 2005 г.*, рабочий релиз - *октябрь 2005 г.*
- *г.*
Sun Microsystems приобрела MySQL AB в 2008 году
- Oracle приобрела *Sun Microsystems 27 января 2010 года*, а Майкл Видениус объявил о форке MySQL, MariaDB
- MySQL Server 5.5 стал общедоступным
- MySQL MySQL Server 6.0.11-альфа был анонсирован *22 мая 2009 г.*

- MySQL Server 8.0 был анонсирован в *апреле 2018 г.*

В настоящее время MySQL развивается и управляется корпорацией Oracle. Позже в этой главе мы обсудим разработанную сообществом версию MySQL — MariaDB.

Установка MySQL в среде Windows

Давайте посмотрим, как установить MySQL в среде Windows. Посетите <https://www.mysql.com> и перейдите на страницу загрузки. MySQL имеет множество редакций. Перейдите на страницу загрузки версии сообщества (<https://dev.mysql.com/downloads/installer/>), поскольку это единственная бесплатная версия MySQL. Ниже приведен скриншот страницы:

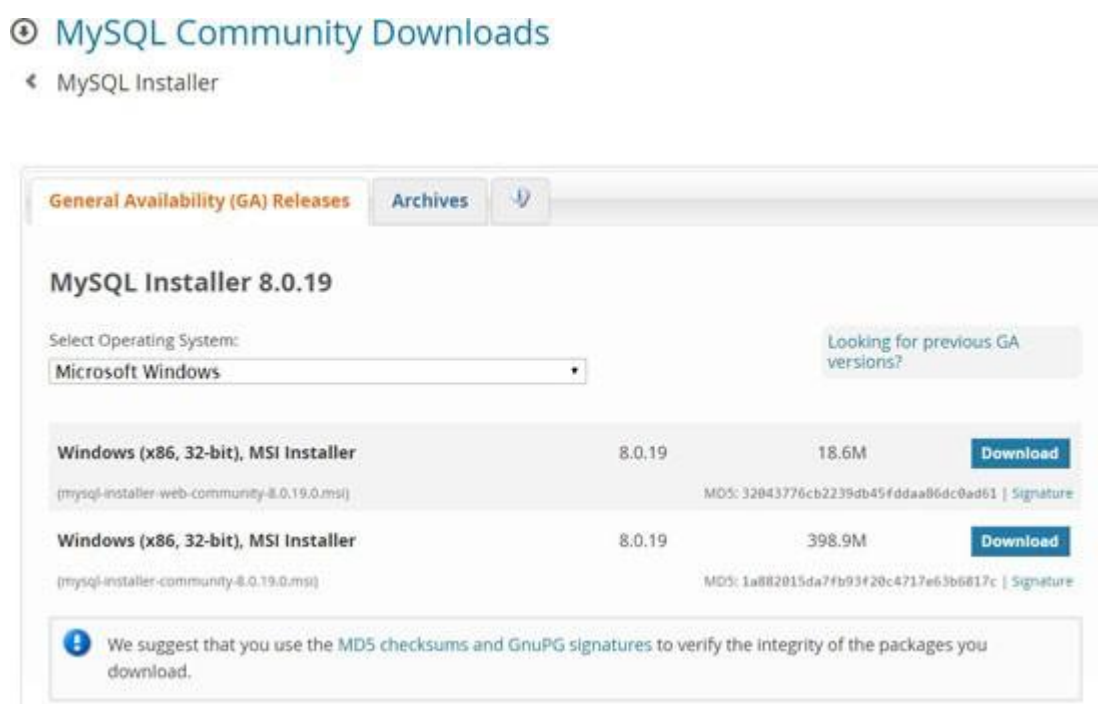


Рис. 1.3

Последняя версия на момент написания книги — 8.0.19. Существует два типа установщиков. Первый — это веб-установщик, который загружает выбранные компоненты из сети Интернет. Именно поэтому его размер меньше. Загрузите его, вы сможете найти его в каталоге Загрузки вашего пользователя. Дважды щелкните, чтобы запустить его. Запуск установки требует прав администратора. Введите логин и пароль администратора, при этом может появиться следующее окно с

сообщением:

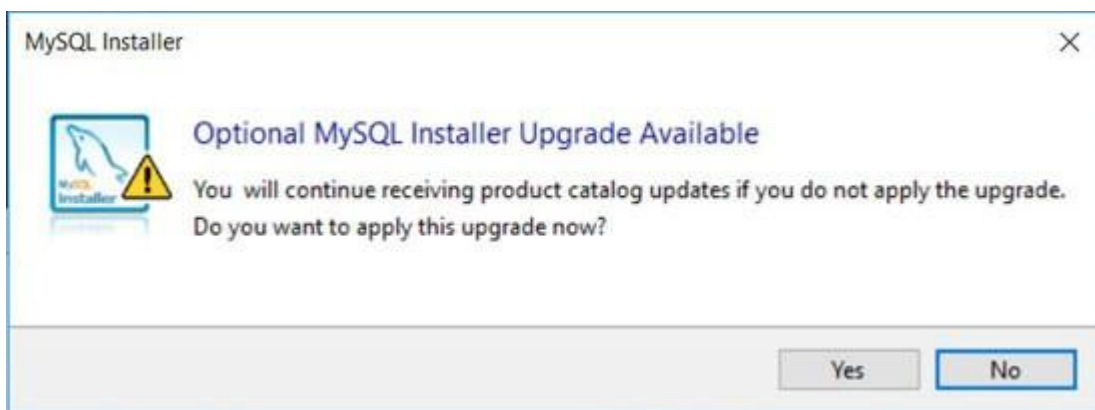


Рис. 1.4

Нажмите кнопку **yes** и обновите установщик. После начала установки выберите вариант **Custom**, как это показано на следующем скриншоте:

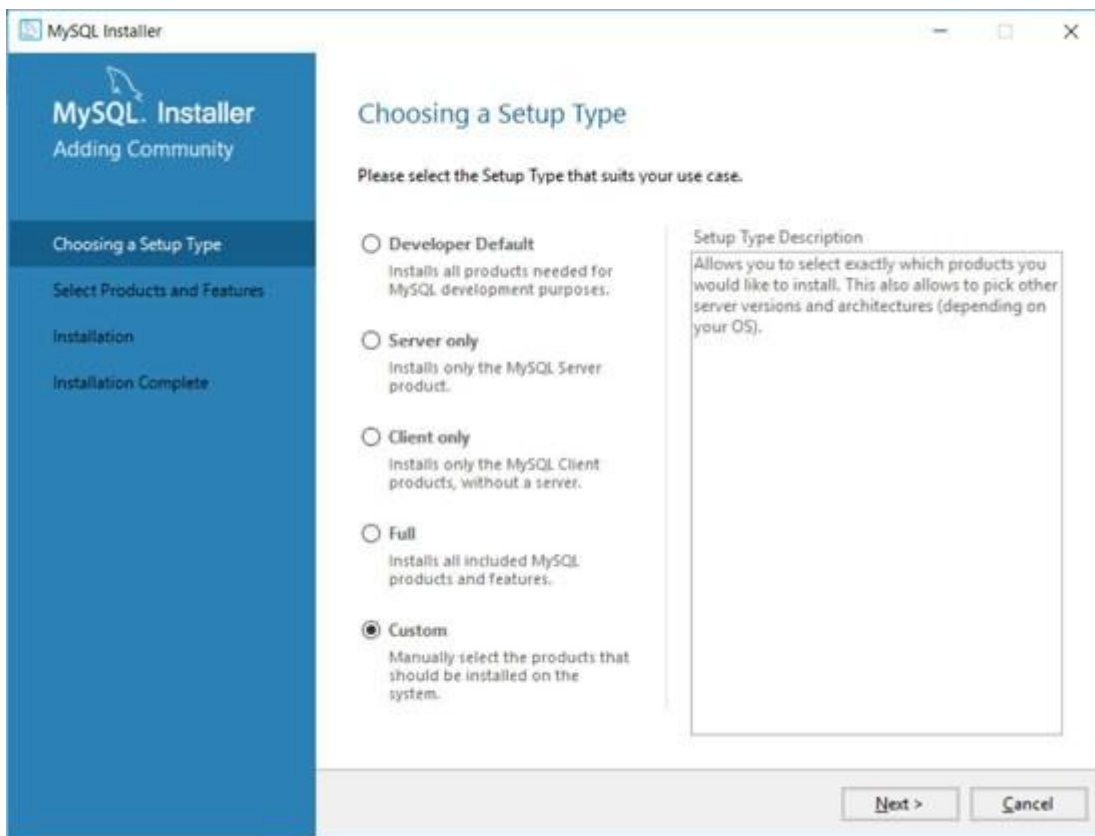


Рис. 1.5

В следующем окне нам будет показан список выбранных компонентов:

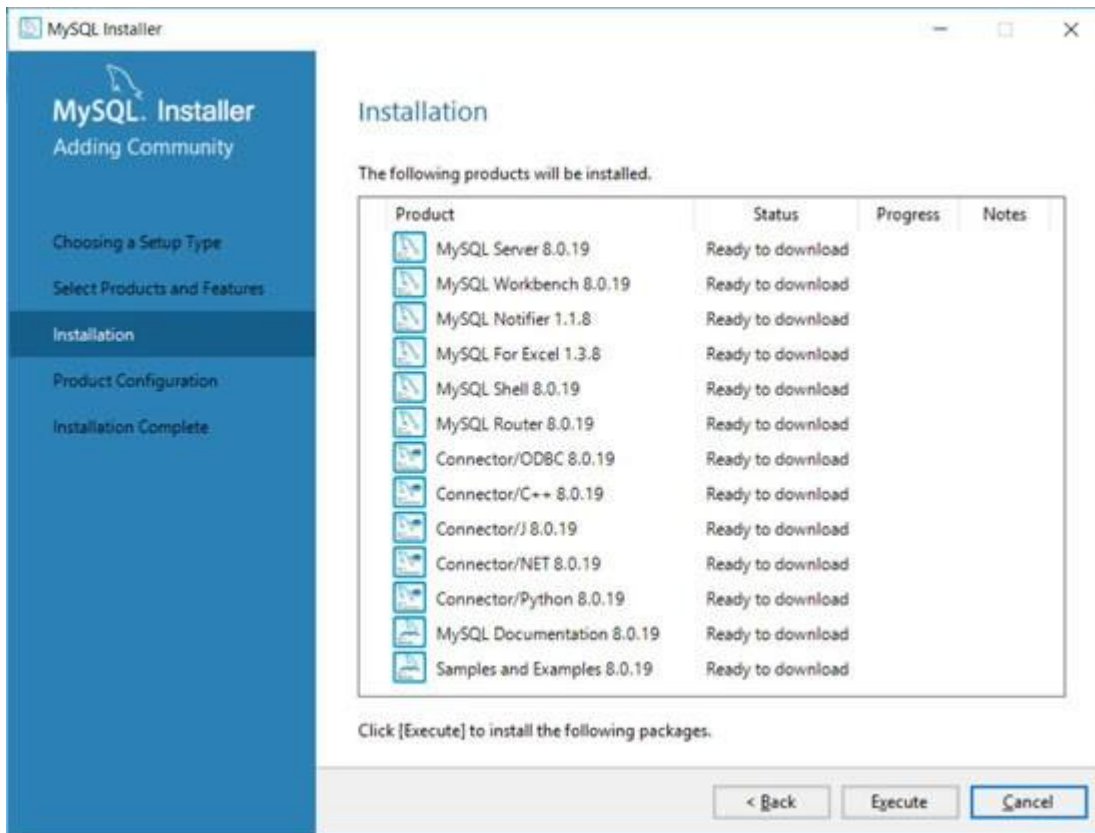


Рис. 1.7

Проверьте список. Если мы что-то пропустили, мы можем перейти к предыдущему окну и добавить компонент, который хотим установить. После проверки списка нажмите кнопку **Execute**. Начнется процесс загрузки и установки, как это показано на следующем скриншоте:

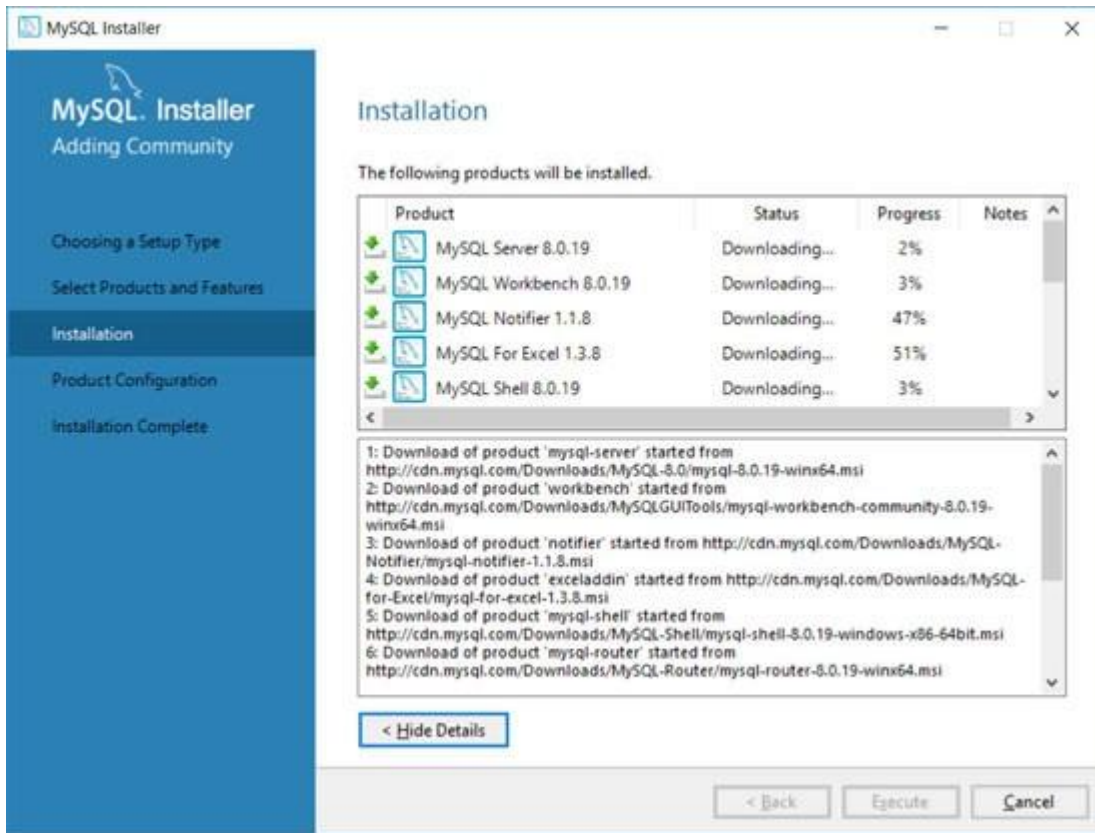


Рис. 1.8

Обратите внимание, что я нажал кнопку **Show Details**. Именно поэтому мастер показывает нам журнал установки, как это показано на скриншоте выше. В зависимости от скорости подключения к сети Интернет загрузка компонентов может занять некоторое время. В случае сбоя загрузки какого-либо компонента будет предложена возможность повторить загрузку этого компонента. После завершения загрузки и установки вместо кнопки **Execute** появится кнопка **Next**. Нажмите на нее, и на экране следующим образом нам будет показан список компонентов, готовых к настройке:

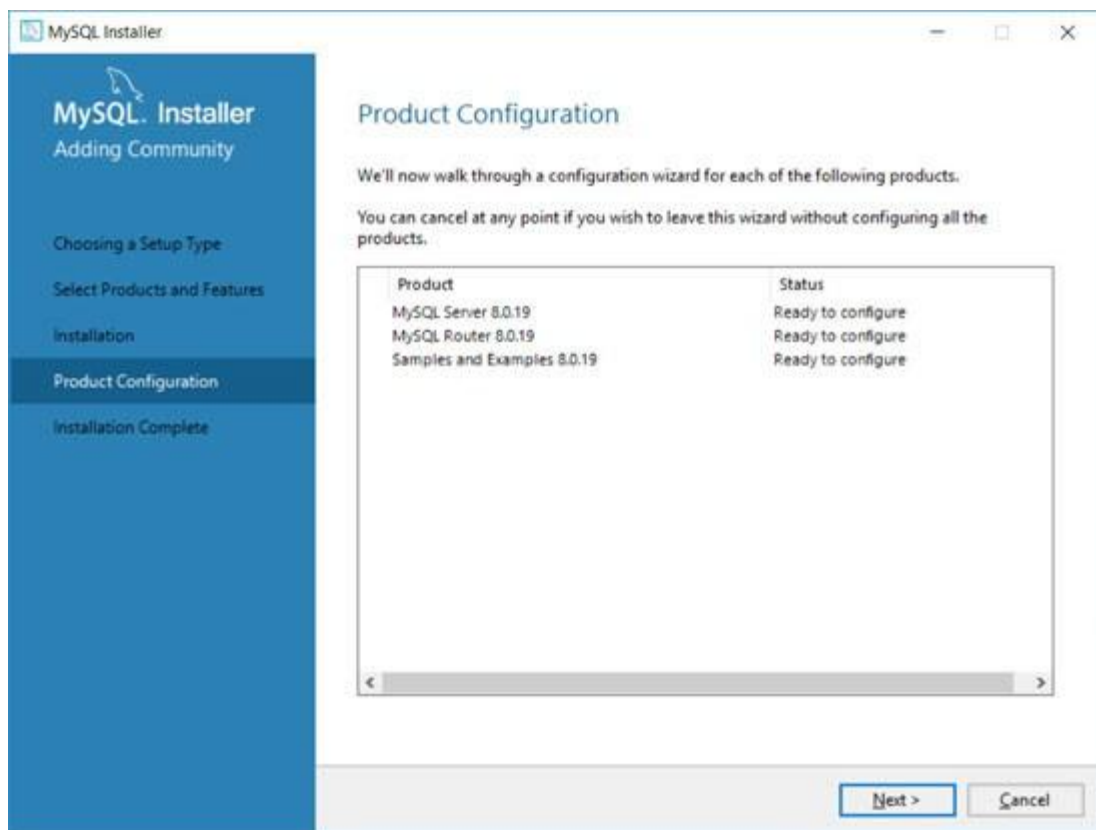


Рис. 1.9

Нажатие на кнопку **Next** запустит мастер настройки компонента MySQL Server. База данных MySQL состоит из множества компонентов. MySQL Server — это компонент, выполняющий задачи СУБД, и его необходимо настроить. Ниже приведен скриншот первого окна мастера настройки **Configuration Wizard**:

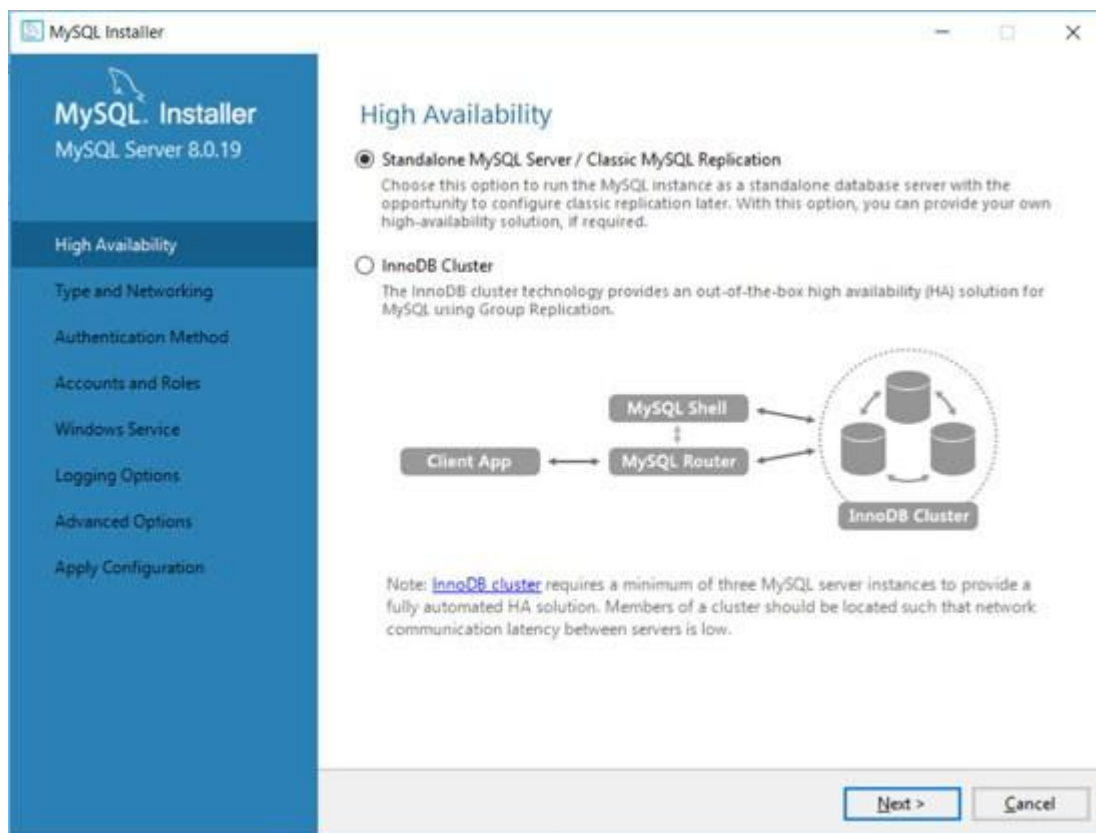


Рис. 1.10

Выберите первый переключатель (Standalone MySQL Server/Classic MySQL Replication), как это показано на скриншоте выше, и нажмите кнопку **next**. На экране следующим образом будут показаны параметры конфигурации:

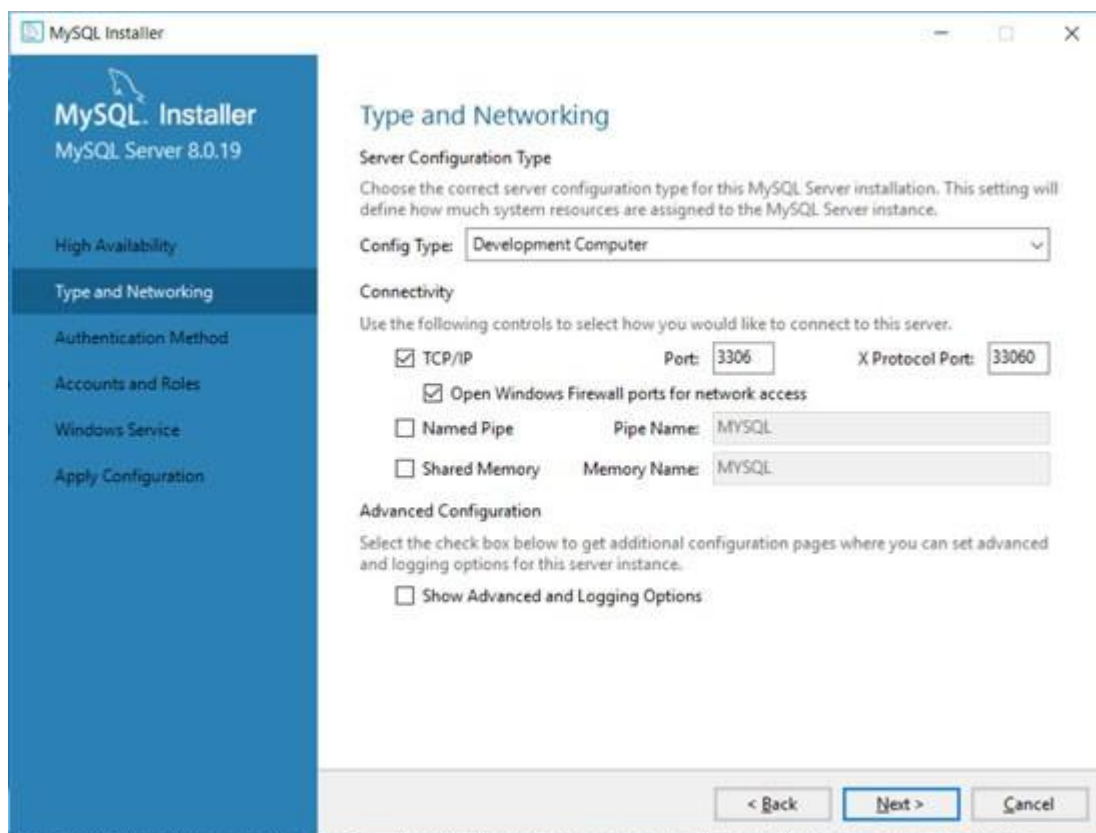


Рис. 1.11

Оставьте все параметры такими, какие они есть, и нажмите кнопку **Next**. Это приведет нас к следующим вариантам:

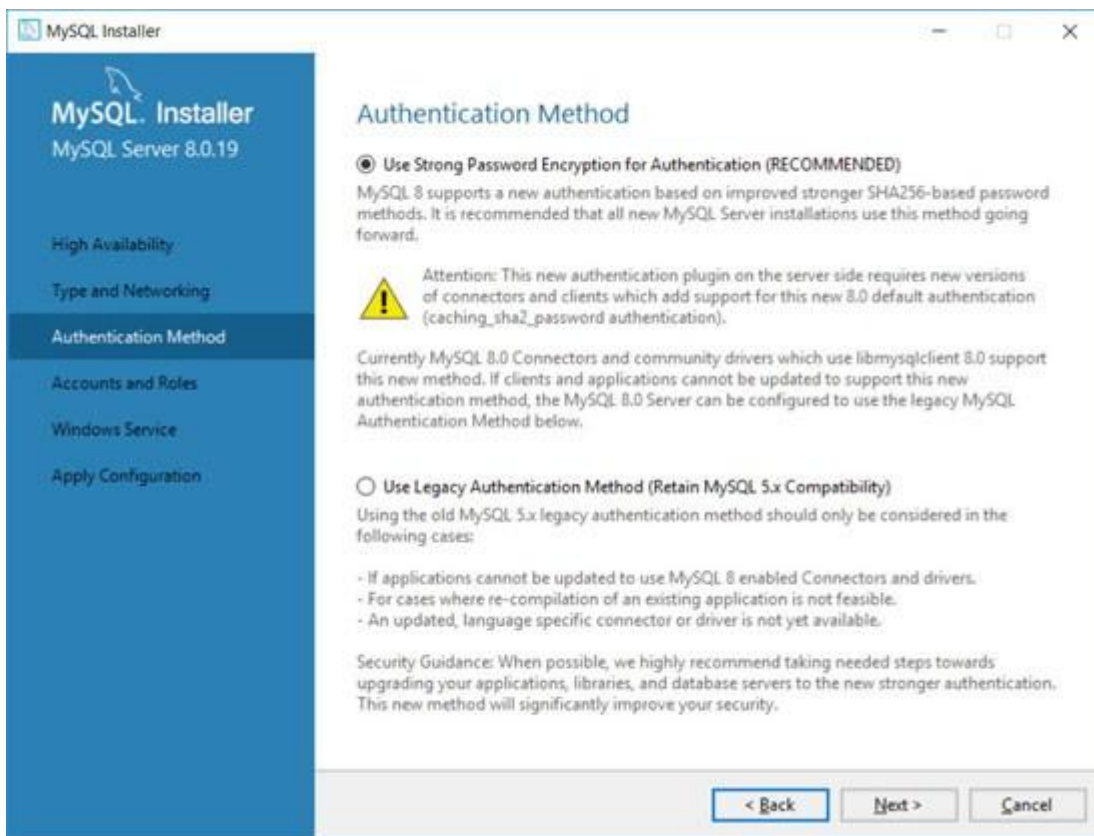


Рис. 1.12

Выберите первый переключатель, это как показано на скриншоте выше, а затем нажмите кнопку **Next**. Это следующим образом приведет нас к настройке учетных записей и ролей:

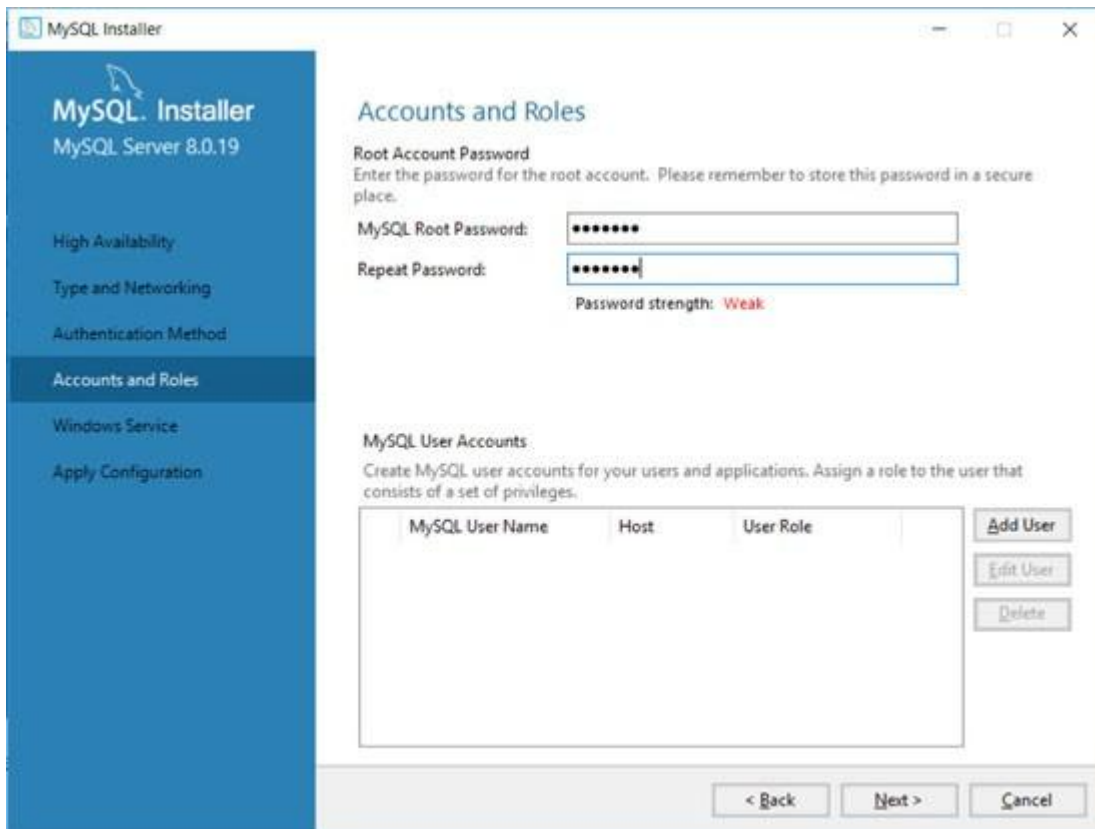


Рис. 1.13

Введите любой пароль по вашему выбору для пользователя root MySQL. Поскольку это учебная база данных, я выбрал простой пароль **test123**. В случае рабочей базы данных пароль должен быть надежным. Нажмите кнопку **Next**, при этом отобразятся следующие параметры:

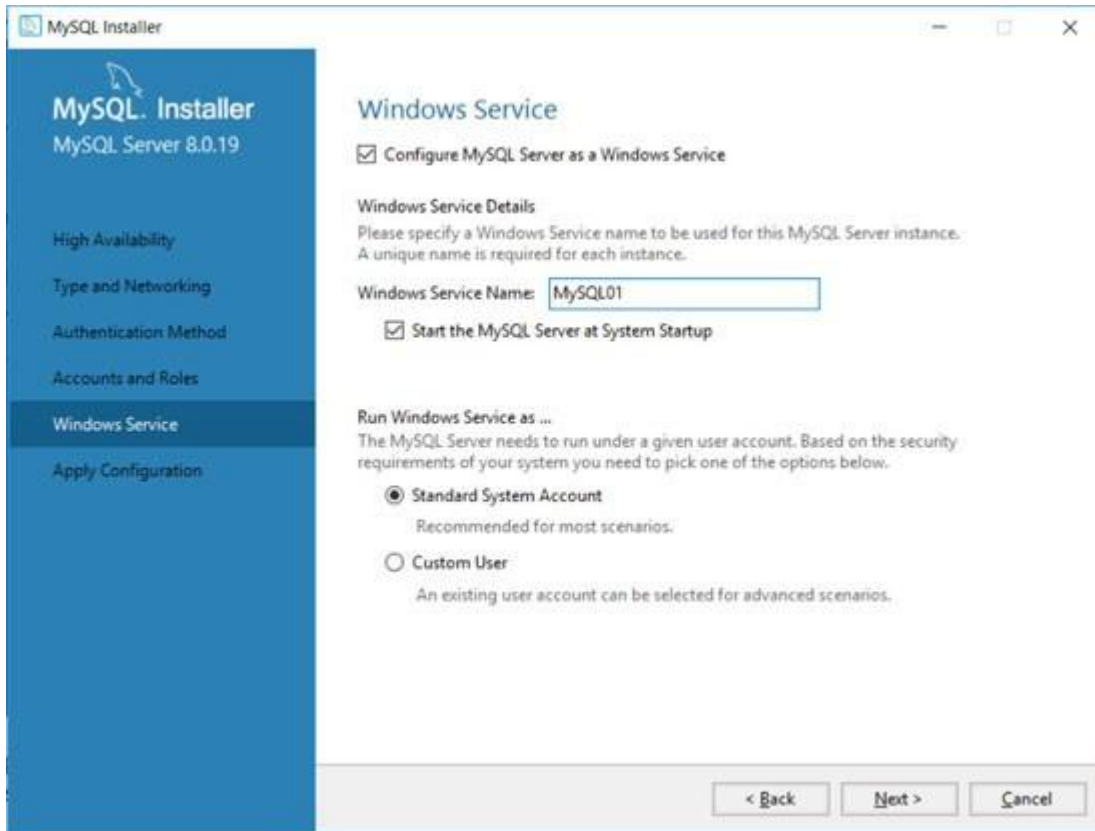


Рис. 1.14

Оставьте все варианты такими, какие они есть. Мы можем изменить строку в текстовом поле **Windows Service Name**. Нажмите кнопку **next**, а в следующем окне нажмите кнопку **execute**. Это запустит Службу, а вы увидите уведомление:

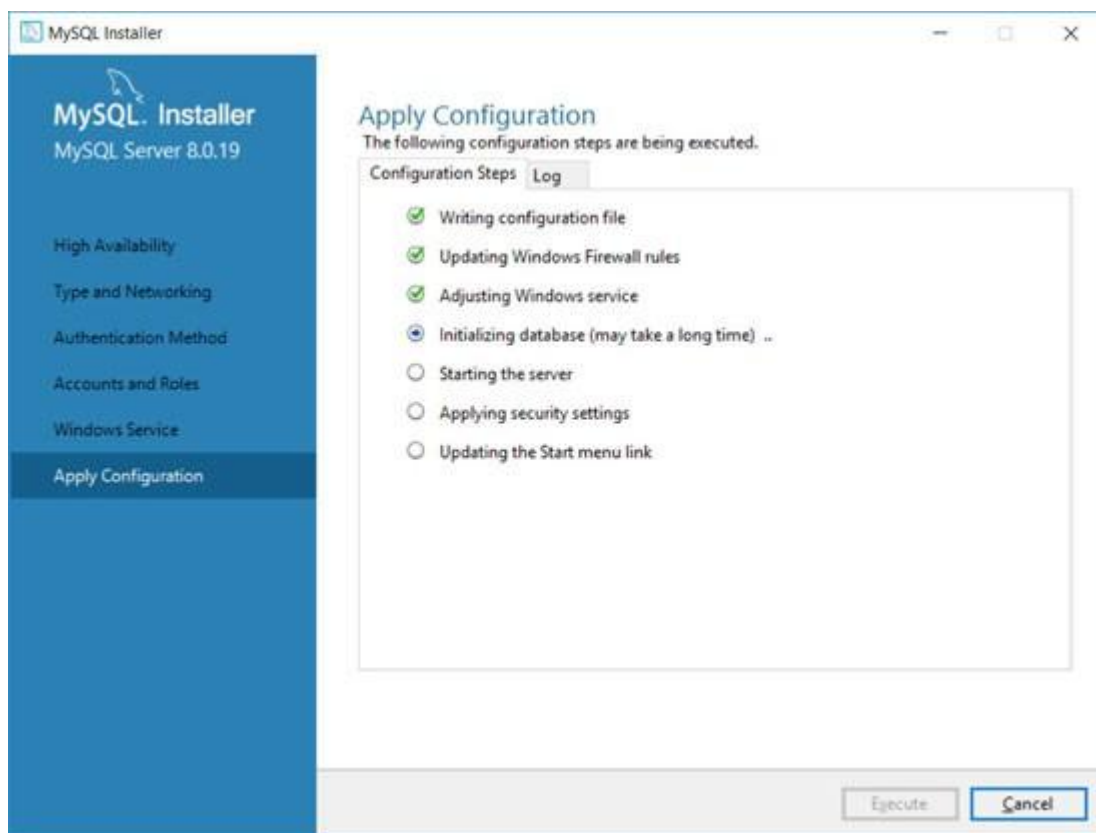


Рис. 1.15

После завершения настройки кнопка **Execute** превратится в кнопку **Finish**. Нажмите на нее, и мы попадем в конфигурацию роутера. На данный момент нам это не нужно, поэтому мы можем просто нажать кнопку **Finish**. Наконец, нам нужно настроить образцы и примеры следующим образом:

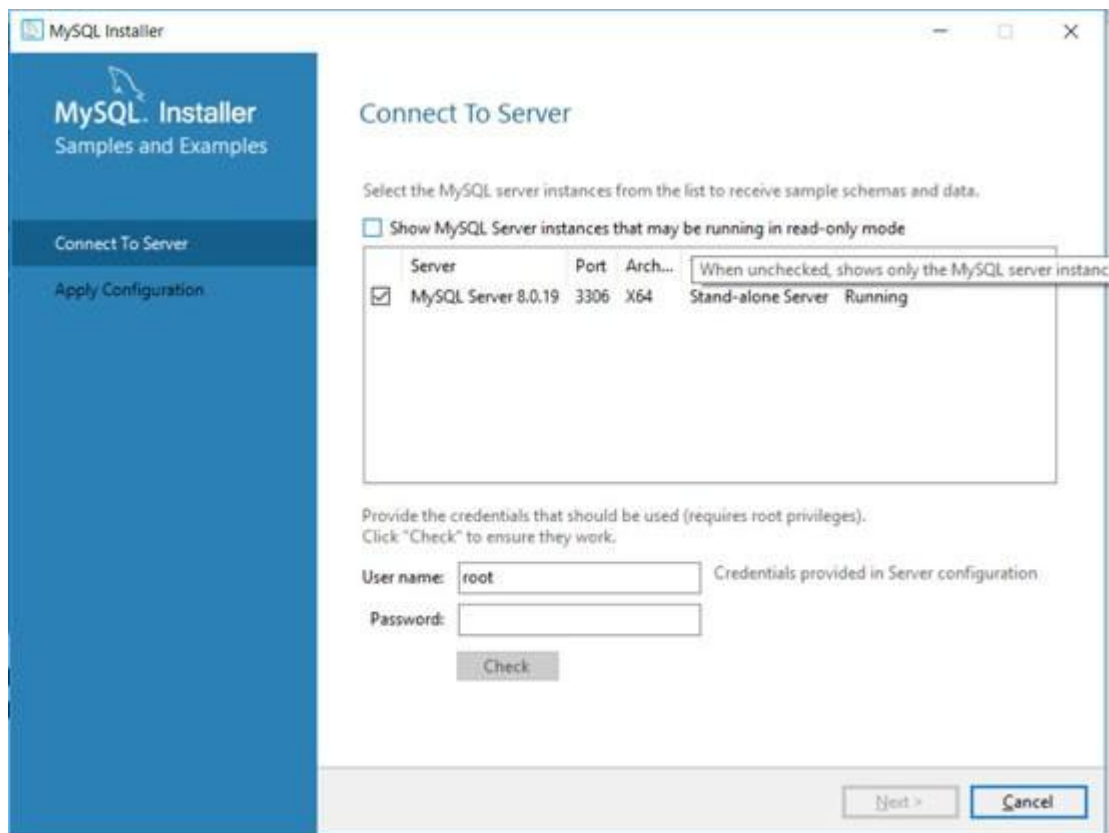


Рис. 1.16

Введите пароль для `root` и нажмите кнопку `check`. После успешного завершения проверки кнопка `next` станет активной. Нажмите на нее, и откроется следующее окно:

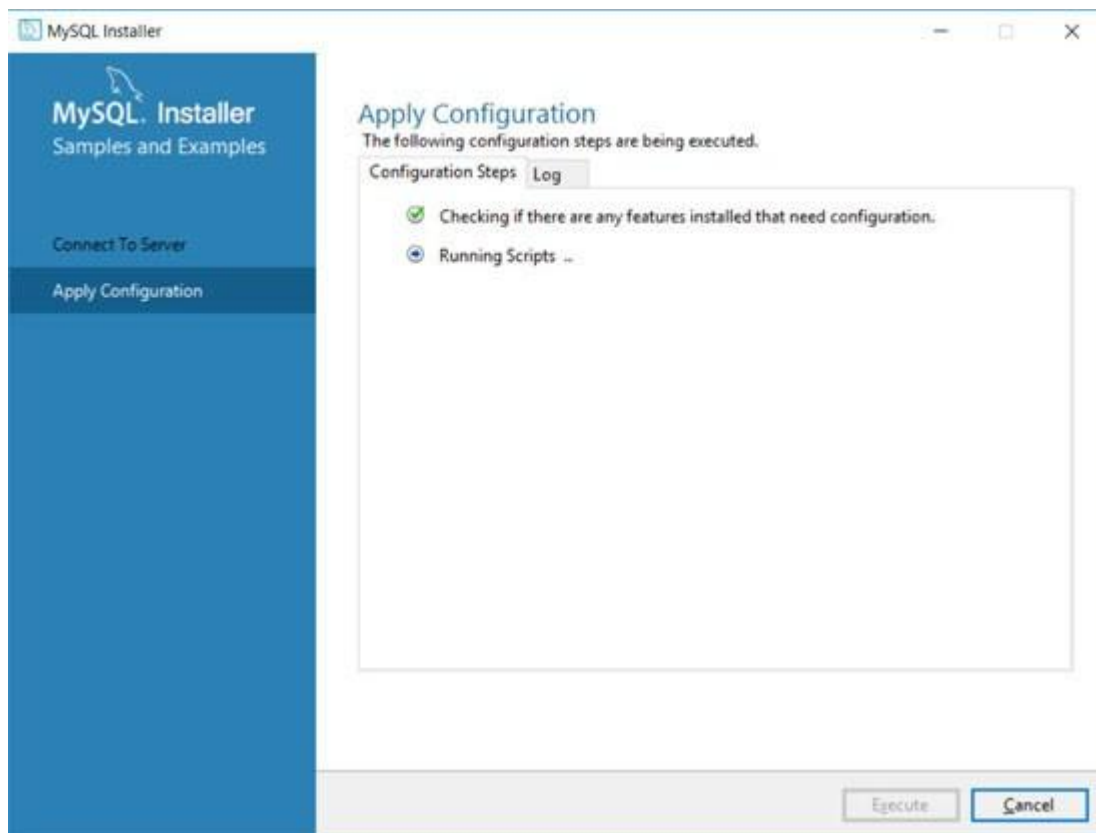


Рис. 1.17

На этом установка в среде Windows завершена. Если нам необходимо перенастроить установку, обновить или установить новые компоненты, мы должны использовать утилиту MySQL Installer, которая устанавливается автоматически при первоначальной настройке. Мы можем найти ее с помощью строки поиска Windows. Ниже приведен скриншот этого:

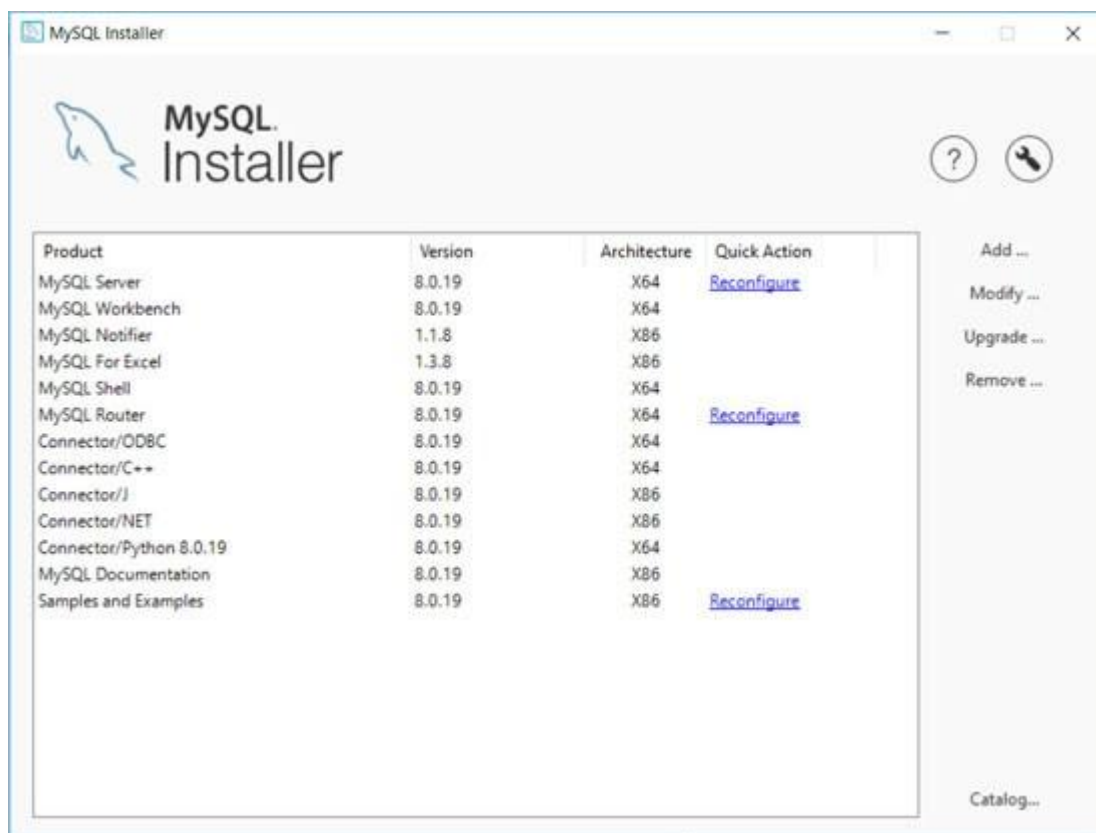


Рис. 1.18

Сервер MySQL запускается автоматически, как только мы запускаем наш компьютер под управлением Windows.

MariaDB

Как объяснялось ранее, MariaDB — это разработанная сообществом версия MySQL. *Мария* — имя младшей дочери *Майкла Видениуса*. MariaDB задумана как бесплатная альтернатива MySQL с открытым исходным кодом и находится под лицензией GNU General Public License. Как и MySQL, MariaDB также используется государственными организациями и другими крупными компаниями. MariaDB и MySQL полностью совместимы друг с другом. Все SQL-запросы MySQL выполняются MariaDB как есть. Даже файлы данных, в которых физические данные хранятся в базе данных, совместимы. Если мы используем MySQL, мы можем просто заменить ее на MariaDB с минимальными изменениями и настройками. Все внешние API и инструменты, поддерживающие MySQL, также поддерживают и MariaDB. Для многих дистрибутивов Linux MariaDB является базой данных по

умолчанию. Более подробную информацию о MariaDB можно найти на <https://mariadb.org/>.

Установка в среде Windows

Давайте посмотрим, как установить MariaDB в Windows. Помните, что мы уже установили экземпляр MySQL в Windows. Загрузите установку с <https://downloads.mariadb.org/>. Ниже приведен скриншот с вариантами загрузки:

MariaDB 10.5.1 Beta 2020-02-14

MariaDB is free and open source software

The MariaDB database server is published as free and open source software under the General Public License version 2. You can download and use it as much as you want free of charge. All use of the binaries from mariadb.org is at your own risk as stated in the GPL v2. While we do our best to make the world's best database software, the MariaDB Foundation does not provide any guarantees and cannot be held liable for any issues you may encounter.

The MariaDB Foundation does not provide any help or support services if you run into troubles while using MariaDB. Support and guarantees are available on commercial terms from multiple MariaDB vendors. There are also many resources you can use to learn MariaDB and support yourself or get peer support online.

Supported and certified binaries available from commercial vendors

These multiple MariaDB vendors that provide

File Name	Package Type	OS / CPU	Size	Meta
Galera 26.4.4 source and packages		Source		
For best results with RPM and DEB packages, use the Repository Configuration Tool.				
mariadb-10.5.1.tar.gz	source tar.gz file	Source	78.1 MB	Checksum Instructions
mariadb-10.5.1-win64-debugsymbols.zip	ZIP file	Windows x86_64	112.0 MB	Checksum Instructions
mariadb-10.5.1-win64.msi	MSI Package	Windows x86_64	57.8 MB	Checksum Instructions
mariadb-10.5.1-win64.zip	ZIP file	Windows x86_64	64.7 MB	Checksum Instructions
mariadb-10.5.1-win32-debugsymbols.zip	ZIP file	Windows x86	86.2 MB	Checksum Instructions
mariadb-10.5.1-win32.msi	MSI Package	Windows x86	53.0 MB	Checksum Instructions
mariadb-10.5.1-win32.zip	ZIP file	Windows x86	59.2 MB	Checksum

Operating System

- DEB Package
- Generic Linux
- RPM Package
- Source Code
- Windows

Package Type

- MacOS pkg
- DEB Package
- RPM Package
- MSI Package
- ZIP file
- source tar.gz file
- source zip file
- grouped tar file

Рис. 1.19

Загрузите соответствующий файл (файл с расширением .msi для Windows) для вашей архитектуры (x64/win32). Мы можем найти его в каталоге Загрузки нашего пользователя. Дважды щелкните, чтобы запустить его. Запуск требует прав администратора. Установка и настройка MariaDB намного проще, чем MySQL. Как только мы запустим установщик, он покажет следующее окно:

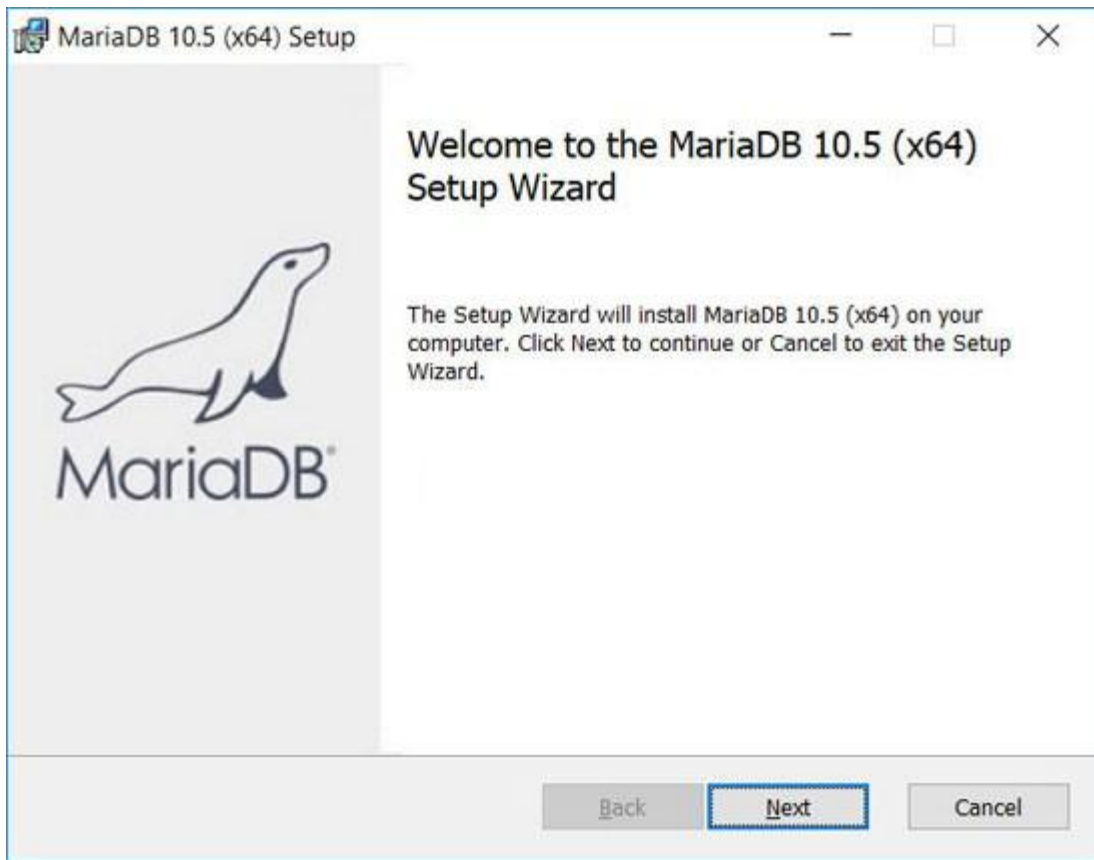


Рис. 1.20

Нажмите кнопку **Next** и откроется следующее окно:



Рис. 1.21

Примите лицензионное соглашение **License Agreement** и нажмите кнопку **Next** здесь, после чего откроется следующее окно:

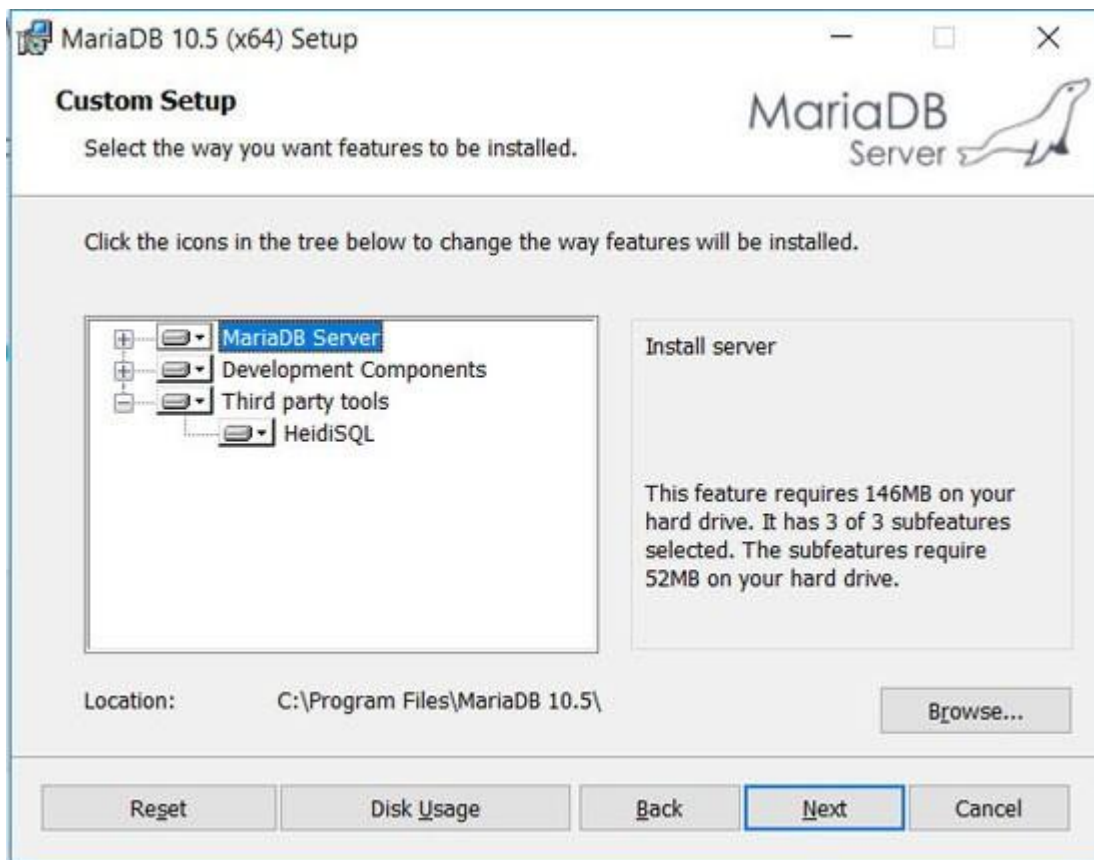


Рис. 1.22

В окне выше мы можем выбрать путь установки. Я рекомендую сохранить параметры по умолчанию, а затем нажать кнопку **next**. Откроется следующее окно:

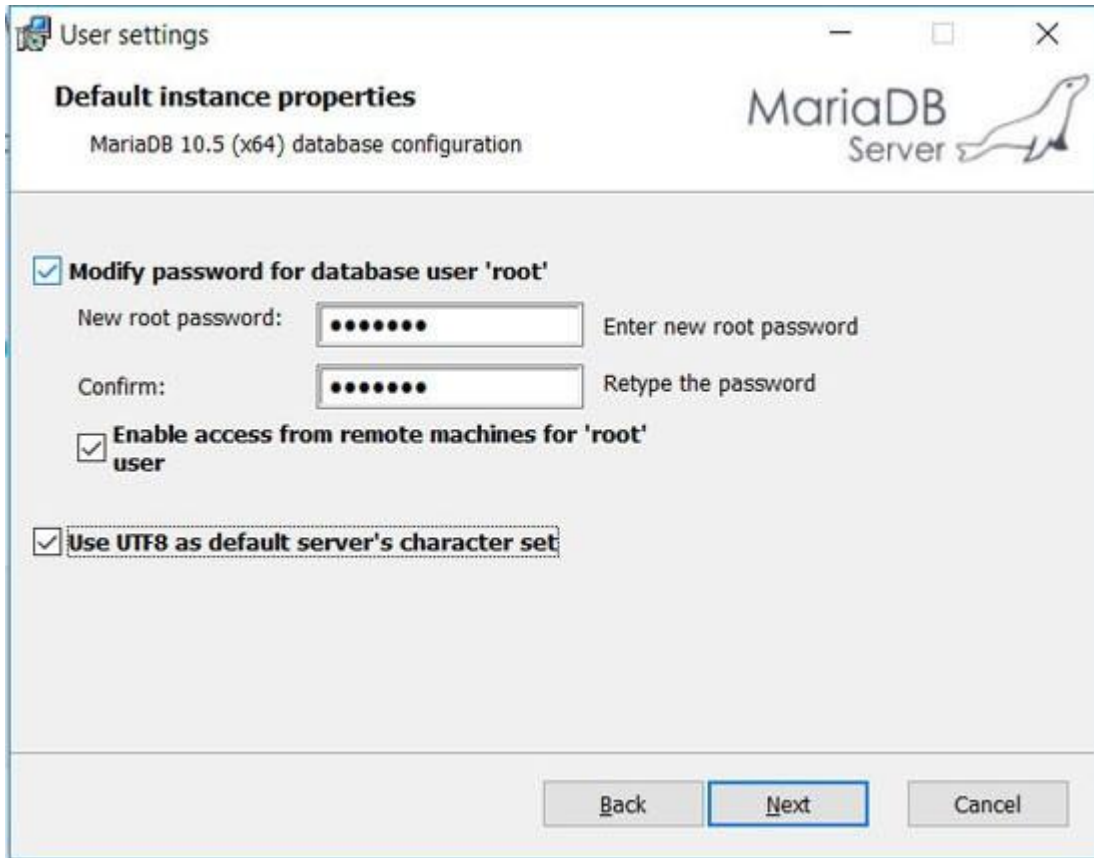


Рис. 1.23

Здесь мы можем установить пароль для учетной записи root. Установите все галочки и нажмите кнопку **Next**. Это откроет следующее окно конфигурирования:

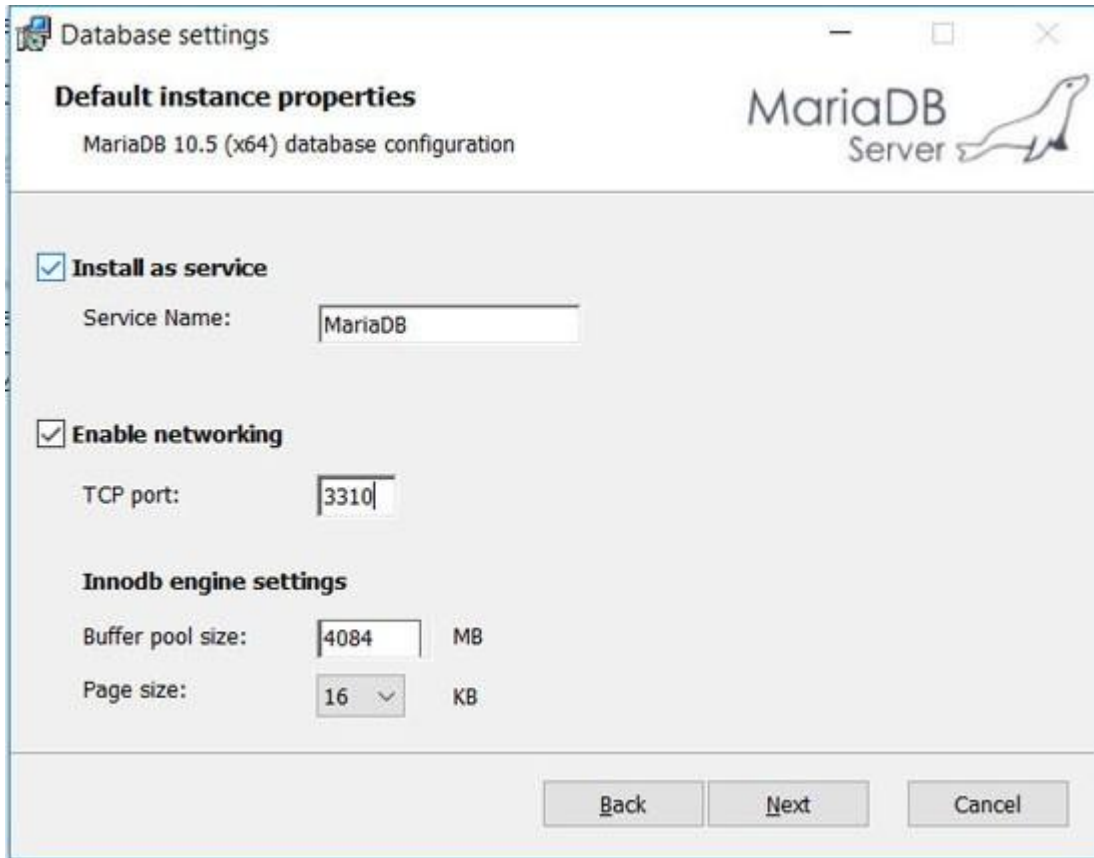


Рис. 1.24

Если вы помните, мы назначили порт с номером 3306 серверу MySQL для сетевого и удаленного подключения. По умолчанию MariaDB тоже использует тот же порт; поэтому просто измените его на 3310, чтобы избежать конфликта. Оставьте остальные параметры такими, какие они есть, и нажмите кнопку **Next**.



Рис. 1.25

Нажмите кнопку **finish**, чтобы завершить процесс установки. Мы установили MariaDB Server и другие утилиты, такие как HeidiSQL. Всякий раз, когда компьютер загружается, автоматически запускается и сервер MariaDB.

[Установка в среде Debian и RaspberryPiRaspbian Linux](#)

Мы можем установить MariaDB в среде Debian и Raspbian. Я использую Raspberry Pi, работающий на Raspbian. MariaDB — это сервер MySQL по умолчанию. Откройте командную строку ОС Raspbian и последовательно выполните следующие команды:

```
sudo apt-get update
sudo apt-get install mariadb-server --fix-missing -y
sudo apt-get install mariadb-client -y
```

Они установят пакеты MariaDB Server и MySQL Client в среде Raspbian/Debian. После завершения установки мы можем выполнить команду `sudo mysql_secure_installation` для настройки сервера MariaDB. Мы можем установить пароль для учетной записи root и ответить «No» на все остальные параметры.

Заключение

В этой главе мы изучили основные понятия, необходимые для начала работы. Мы узнали о проектах MySQL и MariaDB. Мы также установили серверы и клиенты MySQL и MariaDB в среде Windows и Debian/Raspbian Linux. Мы также начали практическое изучение SQL с помощью w3schools. Попробуйте онлайн-редактор SQL.

В следующей главе мы начнем с установки учебных баз данных в MySQL и MariaDB в среде ОС Windows и Raspbian. Мы также увидим, как создать еще одного пользователя для баз данных MySQL и MariaDB.

Что следует помнить

- MariaDB — это среда базы данных MySQL по умолчанию, которая поставляется со многими версиями Linux.
- MariaDB и MySQL глубоко совместимы друг с другом с точки зрения синтаксиса SQL, интерфейсов и инструментов.

Вопросы с выбором правильного ответа

1. Какая из следующих баз данных использует SQL?
 - a) Иерархические базы данных
 - b) Сетевые базы данных
 - c) Реляционные базы данных
 - d) Нереляционные базы данных

Ответы к вопросам

1. a)

Вопросы

1. Что такое база данных?
2. Что такое реляционная база данных?
3. Назовите несколько инструментов, например MySQL Workbench, которые используются для запросов к MySQL и MariaDB.

Ключевые термины

MariaDB, MySQL, Установка, MySQL Workbench, HeidiSQL, SQL, База данных, Реляционная база данных, СУБД, СУРБД, ОРСУБД

ГЛАВА 2

Начало работы с MySQL

https://t.me/it_boooks/2

В предыдущей главе мы изучили несколько основных понятий, связанных с базами данных и СУБД. Мы познакомились с базовыми SQL-запросами с помощью Tryit Online SQL Editor. Мы узнали подробности и историю проектов MySQL и MariaDB. Мы также узнали, как установить MySQL и MariaDB на платформах Windows и Debian/Raspbian. В этой главе мы изучим и продемонстрируем основы MySQL. В этой главе мы не будем начинать работу с полноценными SQL-запросами. Однако мы узнаем, как подключаться к MySQL, как использовать командную строку и графические утилиты, как устанавливать образцы баз данных и как создавать пользователей. Все эти темы нужны, прежде чем мы углубимся в синтаксис SQL.

Структура

В этой главе мы изучим следующие темы:

- Подключение к серверам MySQL и MariaDB разными способами
- Базовые SQL-запросы и установка учебных баз данных

Цель

Основная цель этой главы — подготовить экземпляры сервера MariaDB и MySQL, которые мы установили на платформах Windows и Debian/Raspbian, к демонстрациям. Следование инструкциям этой главы подготовит читателей и настройки к выполнению примеров запросов и упражнений в этой и последующих главах.

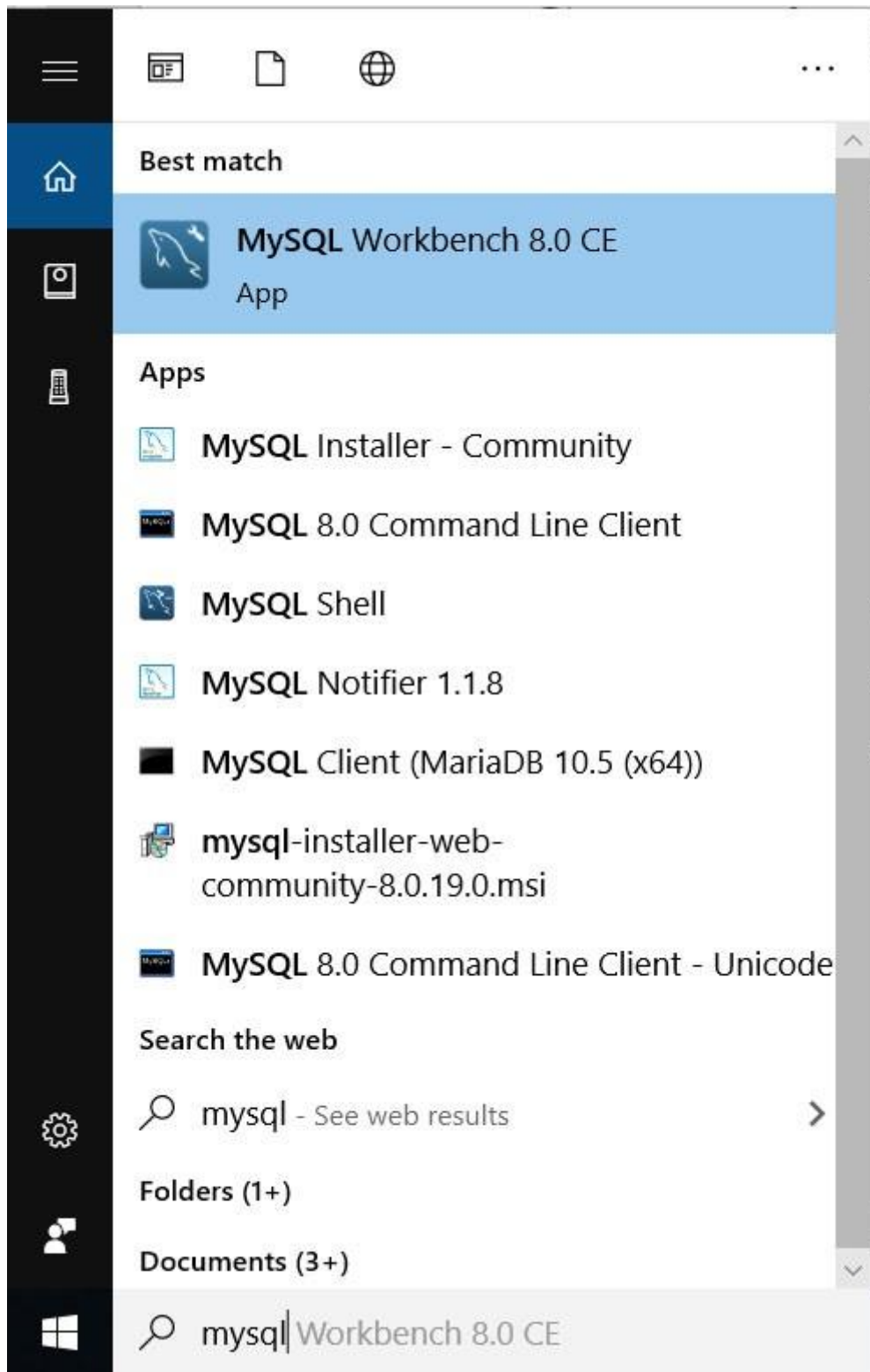
Подключение к экземплярам серверов MySQL и MariaDB

Как мы видели ранее, MySQL и MariaDB имеют серверные компоненты, и их службы запускаются автоматически при каждой загрузке/перезапуске машины.

Эти серверные процессы отвечают за организацию базы данных, физических файлов данных и другие административные задачи. Для работы с базой данных и выполнения SQL-запросов нам необходимо подключиться к этим серверным процессам, используя множество доступных инструментов. В этом разделе мы узнаем, как подключиться к экземпляру сервера MariaDB или MySQL с помощью инструментов, которые предоставляются по умолчанию во время установки.

Соединение с MySQL Workbench

MySQL поставляется с графическим инструментом, известным как MySQL Workbench, который можно использовать для подключения к любому экземпляру MySQL или MariaDB. Мы можем использовать MySQL Workbench для осуществления запросов и основных задач администрирования. Мы можем найти его с помощью меню поиска Windows, выполнив поиск MySQL, как это показано на скриншоте ниже:



Puc. 2.1

Нажмите эту опцию, чтобы запустить **MySQL Workbench**. Ниже приведен скриншот окна рабочей среды MySQL:

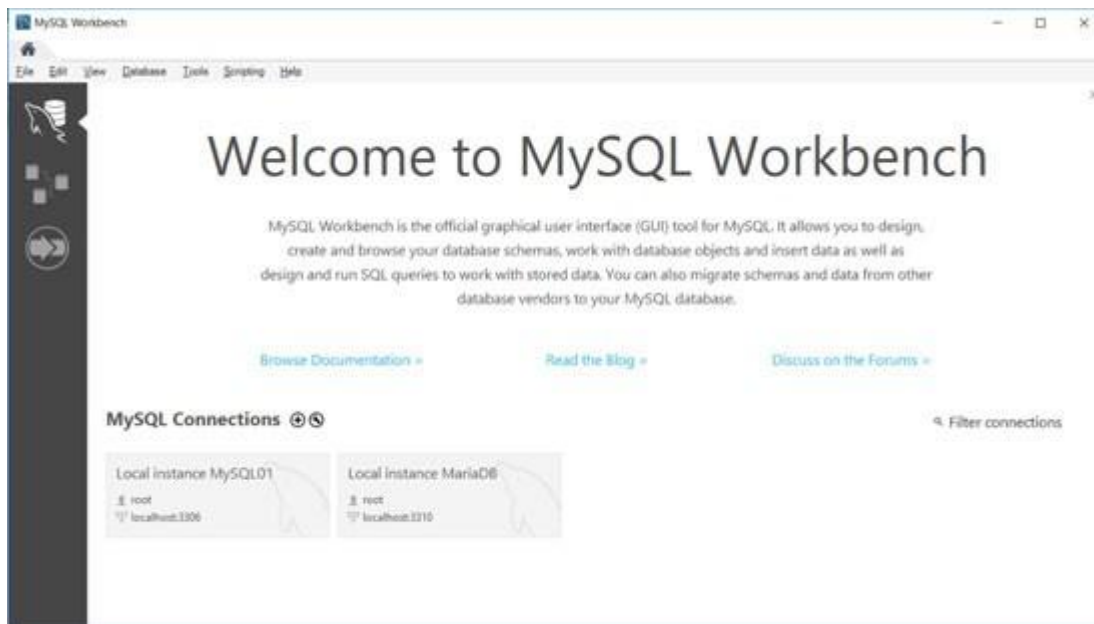


Рис. 2.2

Как мы видим на скриншоте выше, у нас уже есть два доступных соединения. Это связано с тем, что MySQL Workbench обнаружил экземпляры серверных процессов MySQL и MariaDB, запущенные локально (на данном компьютере). Он создал соединение для корневых учетных записей обоих экземпляров сервера, работающих на компьютере. Давайте подключимся к экземпляру сервера MySQL. Дважды щелкните соединение, соответствующее экземпляру сервера MySQL (в данном случае это MySQL01; мы задали это имя во время установки). Откроется диалоговое окно следующего вида:



Рис. 2.3

Введите пароль и установите флажок, если с этого момента вы хотите входить в систему без ввода пароля вручную. После этого откроется следующее окно.

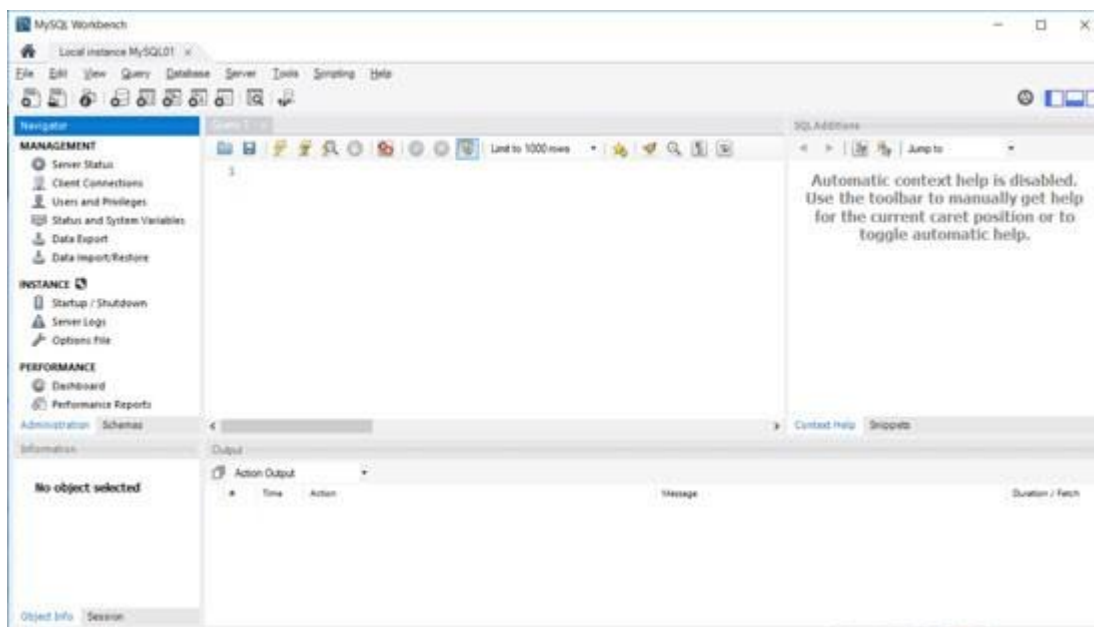


Рис. 2.4

Как мы видим, окно разделено на различные вкладки. На вкладке навигатора у нас есть две опции: **Administration** и **Schemas**. Мы не будем использовать вкладку **Administration** в нашей книге, поскольку она используется для администрирования баз данных, что является отдельной темой, требующей отдельной книги. Нас больше интересуют схемы.

Поэтому нажмите на опцию схем, и вкладка навигатора будет выглядеть следующим образом:



Рис. 2.5

В MySQL (а также в MariaDB) термины «схема» и «база данных» используются как взаимозаменяемые. Если вы до этого работали с другими базами данных, некоторые базы данных, такие как Oracle Database, рассматривают схему как пользователя, а база данных отличается от нашей. В MySQL (и MariaDB) экземпляр сервера может иметь различные базы данных/схемы, а пользователи не привязаны только к одной схеме. Обо всем этом мы узнаем подробно в дальнейшем.

Теперь давайте рассмотрим вкладку **query**. Ниже приведена часть скриншота:

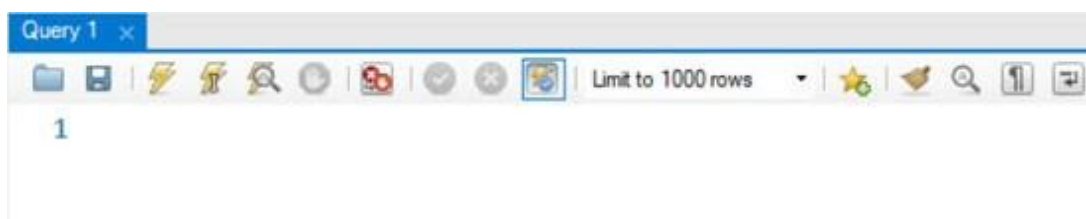


Рис. 2.6

В верхней части мы видим серию значков. Первый значок предназначен для открытия и загрузки сохраненного сценария SQL для выполнения. Мы можем сохранить наши SQL-запросы в виде сценариев на диске, используя значок сохранения (изображение дискеты). Соседний символ молнии предназначен для запуска выбранной части сценария или всего сценария, если фрагмент не выбран. Затем следующий символ молнии со знаком курсора выполняет запрос под текущим положением курсора. Мы можем настроить шрифт текста запроса, одновременно нажав клавиши *Ctrl* и + или -. Вкладка чуть ниже вкладки **query**, здесь мы можем увидеть выходные данные запроса.

Соединение с HeidiSQL

MariaDB поставляется с аналогичным инструментом с графическим интерфейсом. Он известен как HeidiSQL. Мы можем создать новое соединение следующим образом:

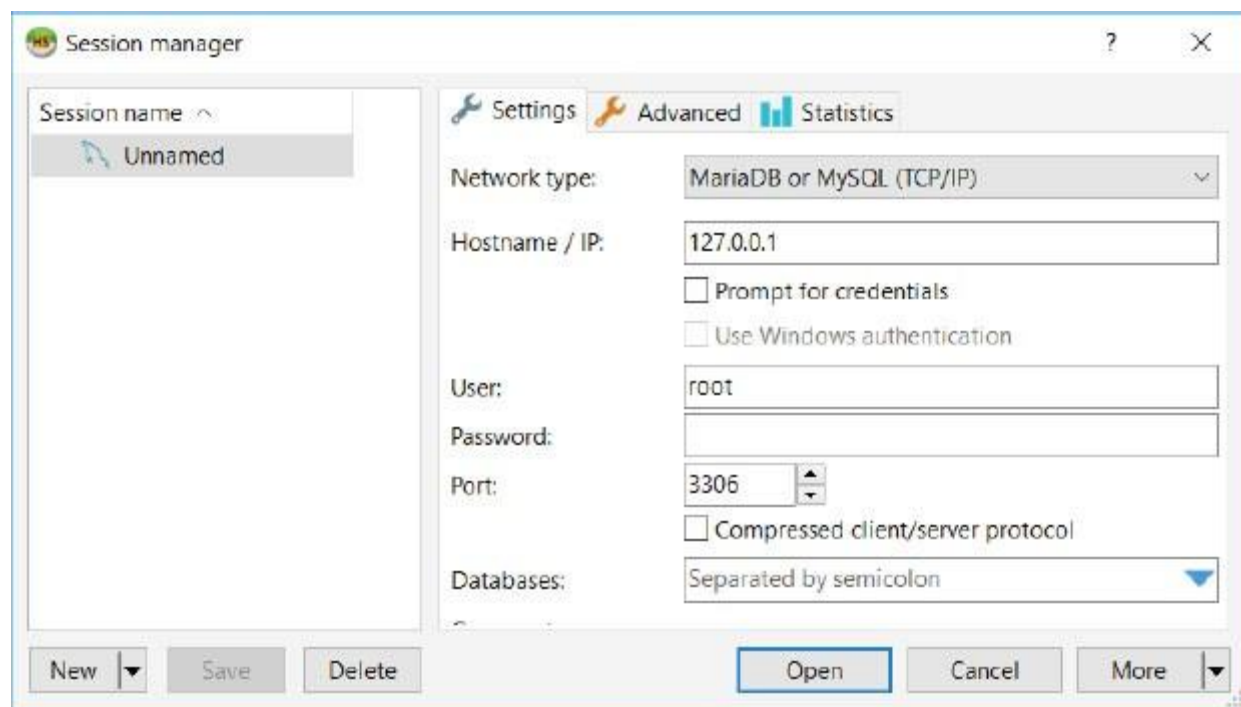


Рис. 2.7

Укажите сведения, относящиеся к экземпляру сервера MariaDB или MySQL, и произойдет подключение. Попробуйте изучить HeidiSQL самостоятельно. Ниже приведен скриншот графического интерфейса HeidiSQL:

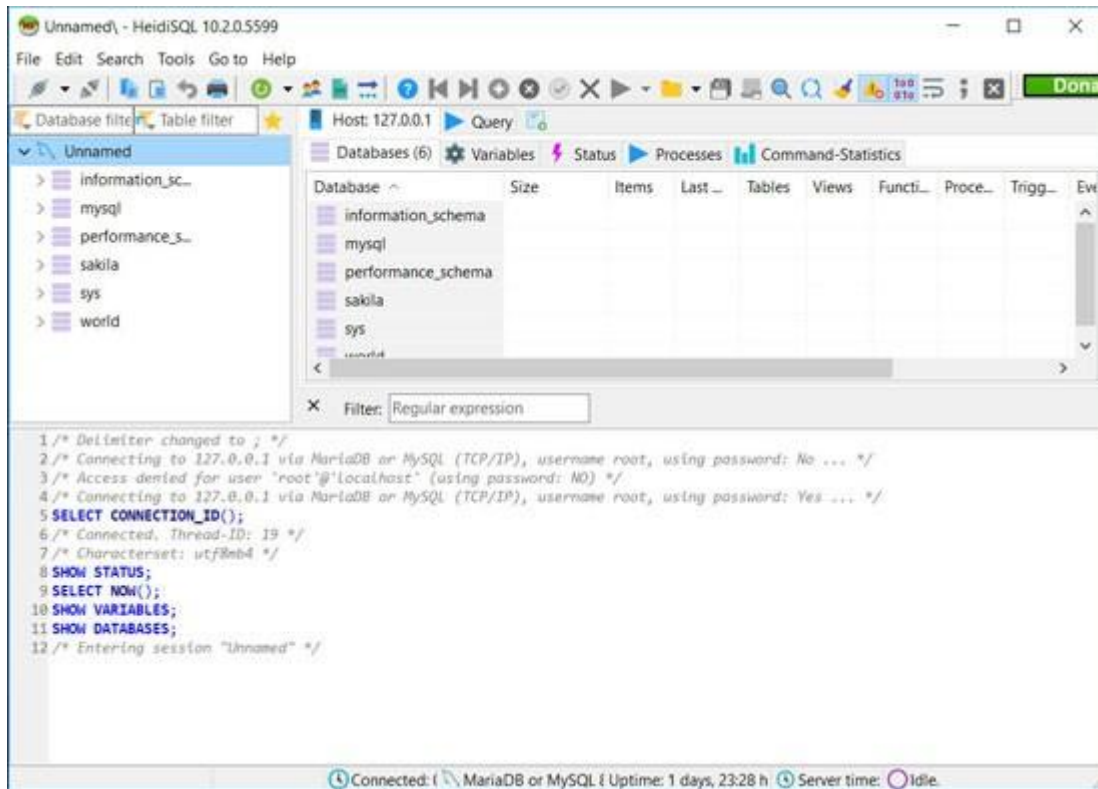


Рис. 2.8

Подключение с помощью командной строки

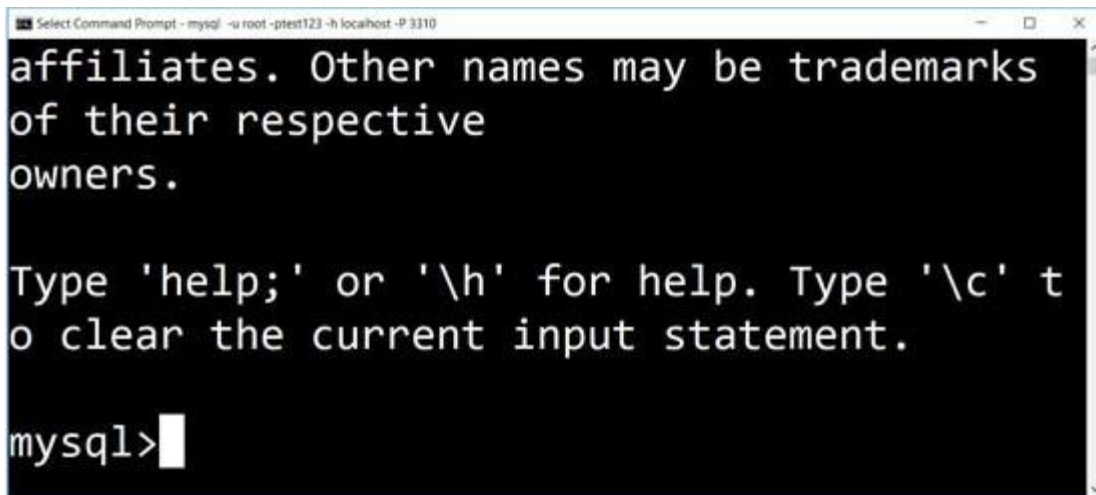
Мы можем подключиться к экземпляру с помощью командной строки Windows. Просто выполните следующую команду в cmd, чтобы подключиться к экземпляру MariaDB:

```
mysql -u root -ptest123 -h localhost -P 3310
```

Чтобы подключиться к экземпляру MySQL:

```
mysql -u root -ptest123 -h localhost -P 3306
```

Между **-p** и паролем нет пробела. Подключение в командной строке выглядит следующим образом:



```
Select Command Prompt - mysql -u root -ptest123 -h localhost -P 3310
affiliates. Other names may be trademarks
of their respective
owners.

Type 'help;' or '\\h' for help. Type '\\c' t
o clear the current input statement.

mysql>
```

Рис. 2.9

ПРИМЕЧАНИЕ. Если выполнение команды завершилась неудачно, и «mysql» не найден, убедитесь, что в переменную среды PATH добавлен путь «C:\Program Files\MySQL\MySQL Server 8.0\bin».

Аналогичным образом мы можем запустить следующую команду на Ixterminal Debian/Raspbian, чтобы подключиться к локальному экземпляру MariaDB:

```
sudo mysql -u root -ptest123 -h localhost -P 3310
```

`sudo` необходимо только тогда, когда мы подключаемся к пользователю root. В противном случае нам это не нужно.

[Подключение к удаленному экземпляру](#)

Мы уже рассмотрели, как подключаться к локальным экземплярам серверов MariaDB и MySQL. Мы даже можем подключаться к удаленным экземплярам с помощью графического интерфейса и командной строки. Все, что нам нужно, это имя пользователя, пароль, имя хоста/IP-адрес и номер порта. Мы можем использовать эти данные для подключения к удаленному серверному процессу MariaDB или MySQL в командной строке или графическом интерфейсе.

[Несколько основных запросов](#)

Мы можем создать нового пользователя с именем пользователя 'testuser' и паролем 'test123', выполнив следующий запрос:


```
CREATE USER 'testuser'@'localhost' IDENTIFIED BY 'test123';
```

Следующий запрос предоставляет все привилегии для всех баз данных/схем вновь созданному пользователю 'testuser':

```
GRANT ALL PRIVILEGES ON *.* TO 'testuser'@'localhost';
```

Мы можем запускать эти запросы в любом интерфейсе (графический интерфейс или командная строка), результат будет одним и тем же;

Если мы используем Debian/Raspbian, мы можем войти в нашу базу данных без использования sudo следующим образом:

```
mysql -u testuser -ptest123 -h localhost -P 3310
```

Теперь, чтобы увидеть, какие образцы схем уже установлены, выполните следующий запрос:

```
SHOW DATABASES;
```

Результат покажет нам несколько систем и несколько примеров баз данных. MySQL поставляется с образцом базы данных sakila. Нам нужно установить ее в MariaDB.

Загрузите zip-файл базы данных с https://github.com/datacharmer/test_db/archive/master.zip, а затем разархивируйте его. Перейдите в каталог с наименованием *sakila* в извлеченном каталоге. Там мы найдем скрипты для создания схемы и наполнения схемы данными. Наименования этих сценариев начинаются со слова *sakila*. Выполните следующие команды в командной строке Windows:

```
mysql -u root -ptest123 -h localhost -P 3310 < sakila-mv-  
schema.sql
```

```
mysql -u root -ptest123 -h localhost -P 3310 < sakila-mv-  
data.sql
```

Run the following commands on the Raspbian / Debian command prompt:

```
sudo mysql -u root -ptest123 -h localhost -P 3310 < sakila-  
mv-schema.sql
```

```
sudo mysql -u root -ptest123 -h localhost -P 3310 < sakila-  
mv-data.sql
```

Нам также необходимо установить еще один пример базы данных,

employees, как в MySQL, так и в MariaDB. Скрипты для установки мы можем найти в родительском каталоге каталога sakila в той же распакованной папке. Наименования этих сценариев начинаются со слова «employees». Выполните следующие команды в командной строке Windows после перехода в этот каталог:

```
mysql -u root -ptest123 -h localhost -P 3310 < employees.sql
```

Запустите следующую команду в командной строке Debian/Raspbian после перехода в каталог, где хранятся сценарии для базы данных *employees*:

```
sudo mysql -u root -ptest123 -h localhost -P 3310 <
employees.sql
```

После завершения установки выполните запрос `SHOW DATABASES` для каждого подключения, чтобы убедиться, что базы данных *sakila* и *employees* отображаются в выходных данных запроса. В любом случае, в инструментрии графического интерфейса мы можем видеть их в браузере схем.

Упражнение

В качестве упражнения к этому разделу выясните, как включить удаленный вход в систему MariaDB, установленную в дистрибутиве Linux Debian/Raspbian.

Заключение

В этой главе мы научились подключаться к экземплярам сервера базы данных с помощью различных инструментов с графическим интерфейсом, таких как MySQL Workbench и HeidiSQL. Мы также узнали, как подключаться с помощью командных строк, таких как `lxterminal` и `cmd`. Затем мы установили образцы баз данных. В ходе этого процесса мы научились запускать сценарии SQL из командной строки. Во всех этих демонстрациях мы изучили, как написать несколько SQL-запросов для выполнения определенных задач с базами данных. Изучив эти темы, мы можем начать изучение SQL с MySQL и MariaDB, начиная со следующей главы.

Что следует помнить

- Установка базы данных не включает в себя множество примеров схем. Мы должны установить их сами.
- Инструменты с графическим интерфейсом, такие как HeidiSQL, Oracle SQL Developer и MySQL Workbench, предоставляют простой в использовании интерфейс для MySQL и MariaDB.

Вопросы с выбором правильного ответа

1. Какая из следующих команд является правильной для отображения списка всех схем базы данных, установленных в текущем экземпляре?
 - a) DISPLAY DATABASES
 - b) LIST DATABASES
 - c) VISUALIZE DATABASES
 - d) SHOW DATABASES

Ответы на вопросы

1. d)

Вопросы

1. Каковы другие бесплатные альтернативы с открытым исходным кодом для MySQL Workbench и HeidiSQL?

Ключевые термины

Подключение, командная строка, HeidiSQL, MySQL Workbench, образцы баз данных, employees, sakila

ГЛАВА 3

Начало работы с SQL-запросами

В предыдущей главе мы узнали, как подключаться к экземплярам серверов MySQL и MariaDB для запуска SQL с помощью командной строки и инструментов графического интерфейса. Мы также установили образец базы данных `employees` и `sakila`, который будет использоваться на протяжении всей книги для демонстрации запросов SQL. В этой главе мы углубимся в синтаксис SQL и изучим несколько основных понятий.

Структура

В этой главе мы изучим следующие темы:

- Начало работы с простыми SQL-запросами
- Выбор и проецирование
- Арифметические операторы и их приоритет
- Псевдонимы столбцов
- NULL
- DISTINCT

Цель

Цель этой главы — начать работу с основными операторами SQL. Прочитав эту главу, читатели смогут выбирать различные схемы и определять разницу между схемами системного каталога и другими схемами, содержащими данные. Читателям также будут понятны операции выбора, проецирования и арифметические операции. Наконец, читатели познакомятся с понятиями `NULL` и `DISTINCT`.

Начало работы с простыми операторами SQL

Давайте начнем с нескольких простых, но важных запросов. Мы уже мельком видели эти запросы ранее. Однако на этот раз мы изучим эти запросы подробно и в соответствующем контексте. Войдите с помощью MySQL Workbench в экземпляр локального сервера MySQL или экземпляр локального сервера MariaDB от имени пользователя root. Мы можем увидеть список всех баз данных (также известных как **схемы**) на вкладке **Schemas** панели **Navigator**, расположенной слева. Если мы дважды щелкнем там на названии базы данных, мы увидим параметры для различных объектов, таких как таблицы и представления. Если мы дважды щелкнем на этих параметрах, мы откроем список соответствующих объектов, доступных пользователю root. Если мы дважды щелкнем по этим объектам, мы увидим более подробную информацию об объектах. Ниже приведен соответствующий скриншот:

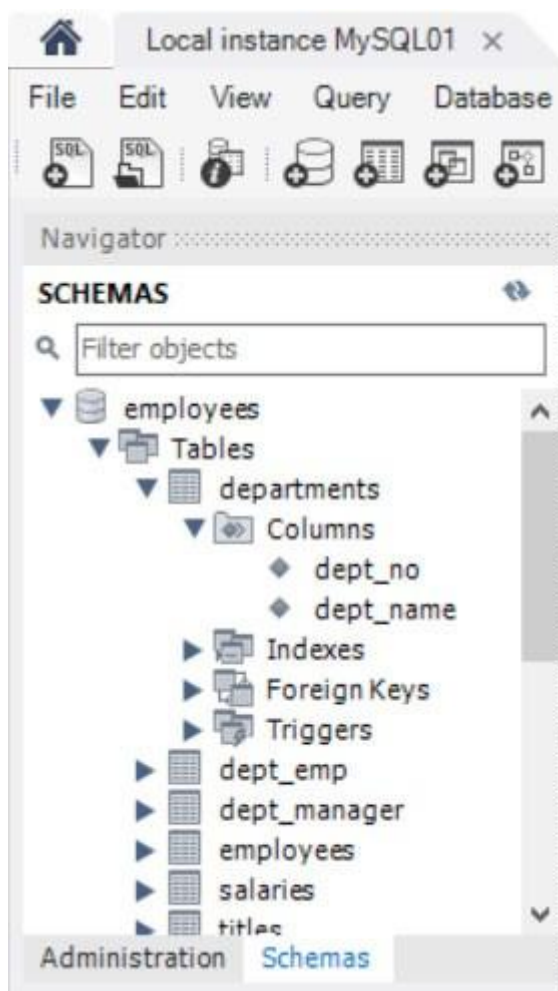


Рис. 3.1

Мы даже можем свернуть эти детали, щелкнув треугольные символы на вкладке. Мы также можем получить список баз данных, выполнив следующую команду или оператор SQL:

```
SHOW DATABASES;
```

В качестве результата мы получим следующие строки:

```
employees  
information_schema  
mysql  
performance_schema  
sakila  
sys  
world
```

Как мы видим в приведенном выше результате, в нем содержится на три схемы больше, чем показано в браузере схем. Эти схемы известны как схемы метаданных, словаря данных или системного каталога. Давайте рассмотрим их один за другим:

- Схема `mysql` — это системная схема. В ней находятся таблицы, содержащие информацию, необходимую серверному процессу MySQL для его выполнения. Она хранит метаданные, относящиеся к объектам и эксплуатационным данным.
- `INFORMATION_SCHEMA` хранит метаданные баз данных. Она хранит информацию о сервере MySQL, такую как имя базы данных и таблицы, тип данных столбца и права доступа.
- `Performance_Schema` отслеживает выполнение процесса MySQL Server на низком уровне. `Performance_Schema` позволяет нам проверять внутреннее выполнение сервера во время выполнения. Основное внимание уделяется данным, связанным с производительностью.

Чтобы запросить такие объекты, как таблицы схемы, нам нужно переключиться на эту схему. Следующий запрос выбирает схему `sakila`:

```
USE SAKILA;
```

Чтобы просмотреть список таблиц в схеме, мы выполняем следующий запрос:

```
SHOW TABLES;
```

Он покажет список таблиц, присутствующих в схеме sakila. Все эти запросы, как обсуждалось ранее, нечувствительны к регистру.

Чтобы выбрать все строки и все столбцы из таблицы ACTOR из схемы, выполните следующий запрос:

```
SELECT * FROM ACTOR;
```

Чтобы увидеть структуру таблицы ACTOR, мы можем выполнить любой из следующих запросов:

```
DESC ACTOR;  
DESCRIBE ACTOR;
```

Это несколько основных, но очень важных запросов, и мы будем регулярно использовать их в демонстрациях на протяжении всей книги.

Выбор и проецирование

Ранее мы видели оператор SELECT. Он выбирает строки в таблице базы данных (или представлении, но сейчас мы сосредоточимся на таблицах). Мы можем ограничить строки с помощью оператора WHERE в запросе SELECT следующим образом:

```
SELECT * FROM ACTOR WHERE FIRST_NAME = 'BURT';
```

Запустите запрос, и вы увидите, что выходные данные ограничены строками, в которых значением столбца FIRST_NAME является BURT. Эта операция выбора определенных строк из таблицы в реляционной терминологии называется **выбором**.

Мы также увидим, что следующий запрос выбирает все строки и столбцы из таблицы:

```
SELECT * FROM ACTOR;
```

Если мы хотим выбрать все строки, но только несколько столбцов, то в реляционной терминологии эта операция называется **проекцией**. Ниже приведен пример:

```
SELECT FIRST_NAME, LAST_NAME FROM ACTOR;
```

Мы даже можем следующим образом изменить порядок столбцов в

операциях проецирования:

```
SELECT LAST_NAME, FIRST_NAME FROM ACTOR;
```

Мы можем следующим образом использовать комбинацию операций выбора и проецирования:

```
SELECT FIRST_NAME, LAST_NAME FROM ACTOR WHERE FIRST_NAME = 'BURT';
```

Арифметические операторы и их приоритет

Мы можем использовать арифметические операции с числовыми столбцами и константами. Когда дело касается приоритета выполнения, эти операторы следуют правилу BODMAS. Реализация SQL MySQL поддерживает следующие операторы:

Оператор	Описание	Пример запроса
+	Оператор сложения	SELECT 10 + 3;
-	Оператор вычитания	SELECT 10 - 3;
*	Оператор умножения	SELECT 10 * 3;
/	Оператор деления	SELECT 10 / 3;
DIV	Оператор целочисленного деления	SELECT 10 DIV 3;
% (или MOD)	Оператор по модулю	SELECT 10 MOD 3; SELECT 10 % 3;

Таблица 3.1

В таблице выше мы видим, что мы используем постоянные числовые значения в операторах `SELECT`. Выполните все запросы один за другим, чтобы понять, как работают эти основные операторы. Мы можем использовать числовые столбцы таблицы и числовые константы в качестве операндов этих арифметических операторов. Ниже приведен простой пример:

```
SELECT title, 12 * rental_rate + 200 FROM film;
```

Выходные данные приведенного выше запроса вычисляются с помощью операций, приоритет выполнения которых подчиняется правилу

BODMAS. По этому правилу умножение вычисляется раньше сложения. Мы можем заставить MySQL сначала произвести сложение, заключив следующим образом операнды в пару простых скобок:

```
SELECT title, 12 * (rental_rate + 200) FROM film;
```

Вот как мы можем работать в качестве операндов с арифметическими операторами с числовыми столбцами таблицы и константами.

Псевдонимы столбцов

Вы, должно быть, заметили, что при отображении результатов запроса отображаются имена столбцов в заголовке. Мы можем выбрать, какими могут быть эти заголовки. Эта концепция известна как **псевдонимы столбцов**. Мы можем указать их после названия столбца. Если в строке, обозначающей псевдоним, есть пробел, то использование двойных кавычек вокруг строки, обозначающей псевдоним, является обязательным. Мы также можем использовать ключевое слово AS между именем столбца и псевдонимом. Следующий запрос представляет все возможные варианты псевдонима:

```
SELECT
title AS "Film Title", release_year "Year of Release",
description AS Summary, original_language_id Language
FROM
film;
```

NULL

NULL — это специальный маркер в базе данных, обозначающий отсутствие какого-либо значения. Это не означает ноль (в случае числовых столбцов) или пустую строку (для символьных или строковых столбцов). Это просто означает, что никакой ценности нет вообще. Его ввел в употребление Э.Ф.Кодд, который первым предложил реляционную модель данных. Это ключевое слово в любой реляционной базе данных. NULL — это состояние, а не значение. Вывод следующего запроса показывает нам NULL в последнем столбце, упомянутом в SELECT:

```
SELECT title, description, original_language_id FROM film;
```

На следующем скриншоте показан результат:

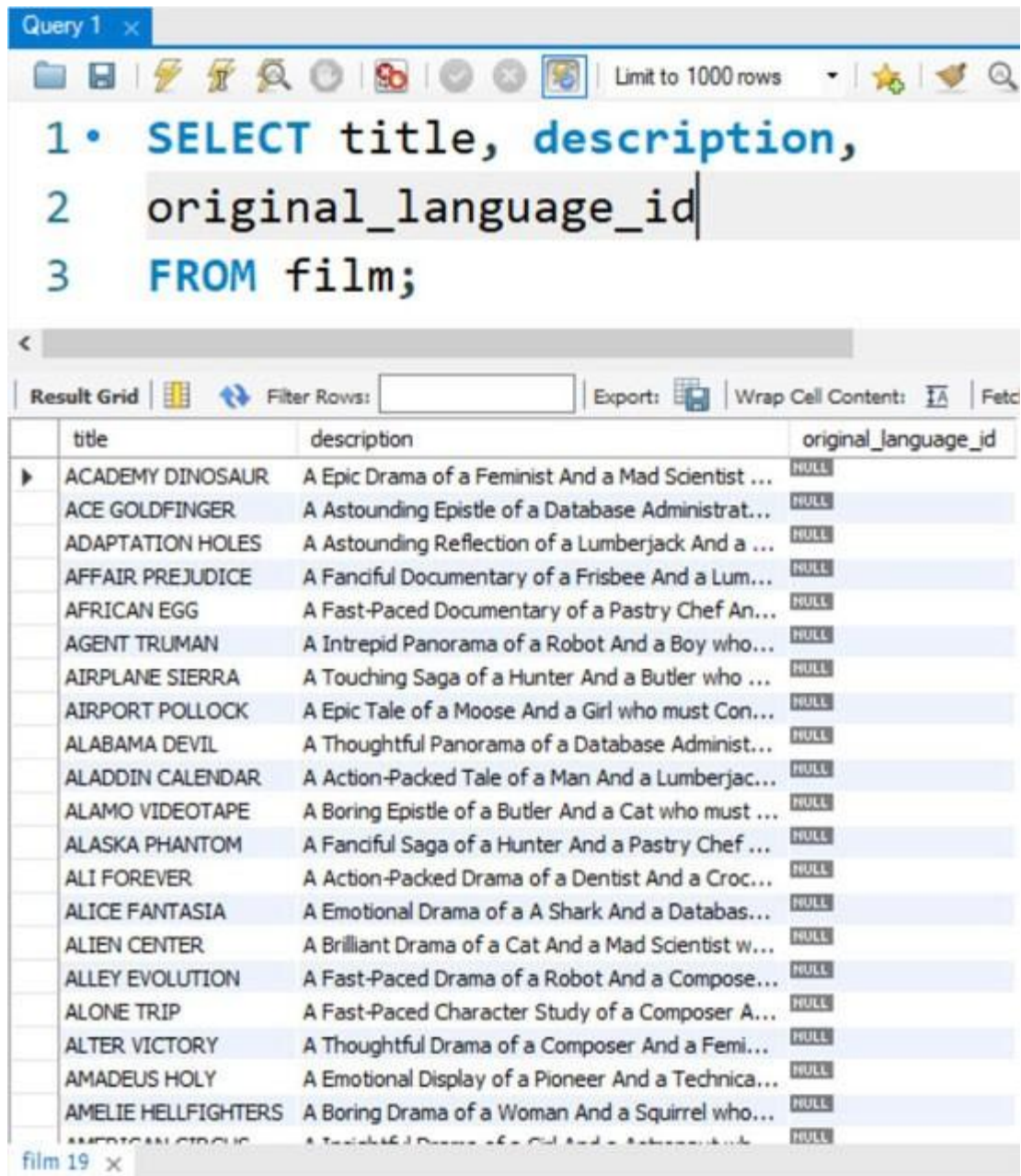


Рис. 3.2

Мы видим NULL при выводе последнего столбца оператора SELECT. В качестве упражнения проверьте, какие другие таблицы хранят NULL.

[DISTINCT](#)

Если мы хотим увидеть отдельные значения для столбца или группы столбцов, мы используем ключевое слово DISTINCT. Результаты также включают NULL. Давайте посмотрим, как он работает шаг за шагом. Запустите следующий запрос, чтобы просмотреть список всех значений в

столбце таблицы:

```
SELECT film_id FROM film_actor;
```

Вы, должно быть, заметили, что значения не уникальны. Чтобы просмотреть список всех уникальных значений, выполните следующий запрос:

```
SELECT DISTINCT film_id FROM film_actor;
```

Давайте рассмотрим, как это работает с комбинацией столбцов. Запустите следующий запрос:

```
select store_id, active from customer;
```

В выведенных результатах мы видим множество повторяющихся пар значений. Мы можем вывести уникальную комбинацию значений для обоих столбцов с помощью следующего запроса:

```
select DISTINCT store_id, active from customer;
```

Мы можем видеть четыре различные комбинации значений в результатах приведенного выше запроса. Мы можем распространить эту логику на более чем два столбца.

Заключение

В этой главе мы начали с базового синтаксиса SQL-запросов для MySQL и MariaDB. Теперь мы знакомы с базовым синтаксисом SQL. Мы изучили и продемонстрировали псевдонимы столбцов, правило BODMAS и арифметические операции. Теперь мы можем писать запросы для выбора уникальной комбинации столбцов и арифметических операций. Мы также увидели значение ключевого слова `NULL` и узнали, как использовать ключевое слово `DISTINCT` для получения уникальной комбинации записей.

В следующей главе мы обсудим различные способы использования оператора `WHERE` в SQL-запросах.

Что следует помнить

- ♦ Арифметические операторы при реализации SQL в MySQL следуют правилам приоритета, определенным BODMAS
- ♦ `NULL` — это состояние, а не значение. Оно означает состояние

отсутствия какого-либо значения

Вопросы с выбором правильного ответа

1. Какая из следующих команд является правильной для выбора схемы с наименованием sakila?
 - a) Choose sakila;
 - b) LIST sakila;
 - c) Show sakila;
 - d) Use sakila;
2. Значение NULL это:
 - a) Пустая строка
 - b) Ноль
 - c) Отсутствие какого-либо значения
 - d) бесконечность
3. Какое ключевое слово следует использовать, чтобы выбрать уникальную комбинацию значений из таблицы?
 - a) UNIQUE
 - b) DISTINCT
 - c) SORT
 - d) DIFFERENT

Ответы на Вопросы

1. d)
2. c)
3. b)

Вопросы

1. Как получить уникальные значения для столбца или комбинации столбцов из таблицы базы данных?
2. Объясните значение ключевого слова NULL.
3. Что такое метаданные? Каковы схемы метаданных в MySQL и MariaDB?

4. Укажите разницу между операциями выбора и проецирования.

Ключевые термины

Выбор, Проекция, NULL, DISTINCT, Операторы, Приоритет

ГЛАВА 4

Оператор WHERE в подробностях

В предыдущей главе мы изучили и продемонстрировали базовый синтаксис SQL, а также несколько относительно простых концепций и их демонстрацию с помощью запросов SQL. Теперь мы знаем как подключаться к экземплярам MySQL и MariaDB, а также выполнять простые запросы.

В этой главе мы сосредоточимся на операции выбора и подробно изучим тонкости оператора `WHERE`.

Структура

Ниже приведен подробный список тем, которые мы изучим и продемонстрируем с помощью SQL-запросов в этой главе:

- Оператор `WHERE`
- Операторы сравнения
- Оператор `LIKE`
- Сравнение с `NULL`
- Логические операции
- Сортировка набора результатов
- Работа с датами

Цель

The objective of this chapter is to make readers comfortable with the selection operation. After following this chapter, the readers will be able to effectively use the `WHERE` clause in the SQL queries to filter the data. This chapter forms base for many types of queries we will learn throughout the coming chapters in the book. For example, we can use the `WHERE` clause in the SQL queries about the join operation.

Оператор WHERE

Давайте рассмотрим, как используется оператор `WHERE`. Мы можем использовать его для фильтрации набора результатов, возвращаемого запросом `SELECT`. Давайте рассмотрим, как фильтровать данные. Запустите следующие запросы:

```
USE sakila;
```

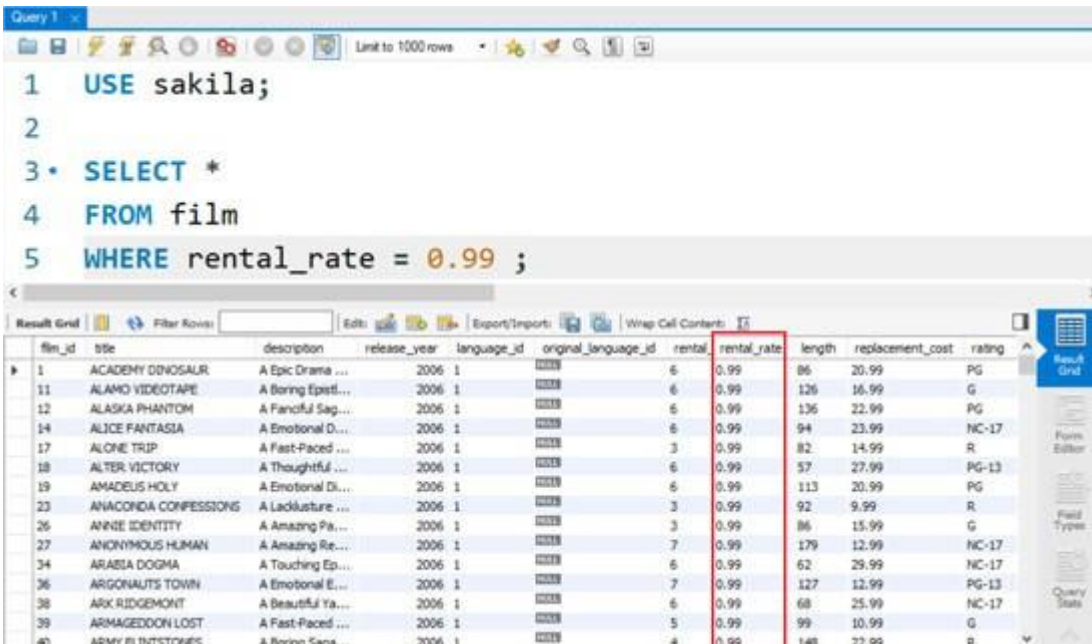
Теперь давайте напишем простой запрос `SELECT`:

```
SELECT * FROM film;
```

Этот запрос, как мы уже знаем, возвращает все строки из таблицы `film`. Мы можем фильтровать возвращаемые строки на основе выбранного нами критерия, используя оператор `WHERE`. Ниже приведен пример:

```
SELECT * FROM film WHERE rental_rate = 0.99 ;
```

Этот запрос возвращает все строки, в которых значение столбца `rental_rate` равно `0.99`. На следующем скриншоте показаны запрос и выходные данные:



film_id	title	description	release_year	language_id	original_language_id	rental	rental_rate	length	replacement_cost	rating
1	ACADEMY DINOSAUR	A Epic Drama ...	2006	1	ENGLISH	6	0.99	86	20.99	PG
11	ALAMO VIDEOTAPE	A Boring Epi...	2006	1	ENGLISH	6	0.99	126	16.99	G
12	ALASKA PHANTOM	A Fantastic Sag...	2006	1	ENGLISH	6	0.99	136	22.99	PG
14	ALICE FANTASIA	A Emotional D...	2006	1	ENGLISH	6	0.99	94	23.99	NC-17
17	ALONE TRIP	A Fast-Paced ...	2006	1	ENGLISH	3	0.99	82	14.99	R
18	ALTER VICTORY	A Thoughtful ...	2006	1	ENGLISH	6	0.99	57	27.99	PG-13
19	AMADEUS HOLY	A Emotional Di...	2006	1	ENGLISH	6	0.99	113	20.99	PG
23	ANACONDA CONFESSIONS	A Lacklustre ...	2006	1	ENGLISH	3	0.99	92	9.99	R
26	ANNE IDENTITY	A Amazing Pa...	2006	1	ENGLISH	3	0.99	86	15.99	G
27	ANONYMOUS HUMAN	A Amazing Re...	2006	1	ENGLISH	7	0.99	179	12.99	NC-17
34	ARABIA DOGMA	A Touching Ep...	2006	1	ENGLISH	6	0.99	62	29.99	NC-17
36	ARGONAUTS TOWN	A Emotional E...	2006	1	ENGLISH	7	0.99	127	12.99	PG-13
38	ARK RIDGEMONT	A Beautiful Ya...	2006	1	ENGLISH	6	0.99	68	25.99	NC-17
39	ARMAGEDDON LOST	A Fast-Paced ...	2006	1	ENGLISH	5	0.99	99	10.99	G
40	ARMY FLINTSTONES	A Boring Sag...	2006	1	ENGLISH	4	0.99	148	22.99	R

Рис. 4.1

Мы видим, что все строки в выводе выше соответствуют условию фильтра. Это пример числовых данных. Мы также можем использовать оператор `WHERE` следующим образом для фильтрации символьных

строк:

```
SELECT * FROM film WHERE title = 'ALONE TRIP';
```

Этот запрос возвращает одну строку, в которой название фильма соответствует строке, указанной в операторе запроса `WHERE`.

Операторы сравнения

Мы можем использовать операторы сравнения в выражении `WHERE` запроса `SELECT`. Давайте рассмотрим пример:

```
SELECT * FROM film WHERE rental_rate > 1.99;
```

В этом запросе использовался оператор сравнения «больше чем». Аналогичным образом мы можем использовать другие операторы сравнения следующим образом:

```
SELECT * FROM film WHERE rental_rate >= 1.99;
```

```
SELECT * FROM film WHERE rental_rate < 1.99;
```

```
SELECT * FROM film WHERE rental_rate <= 1.99;
```

У нас также есть оператор `Not Equal To` («Не равно»). Его можно записать двумя способами:

```
SELECT * FROM film WHERE rental_rate <> 2.99;
```

```
SELECT * FROM film WHERE rental_rate != 2.99;
```

Мы также можем сравнить данные столбца с числовым диапазоном, используя ключевые слова `BETWEEN` и `AND` следующим образом:

```
SELECT title, rental_rate FROM film WHERE rental_rate BETWEEN  
0.99 AND 2.99;
```

Приведенный выше запрос следующим образом возвращает все данные, попадающие в заданный диапазон (включая 0.99 и 2.99):

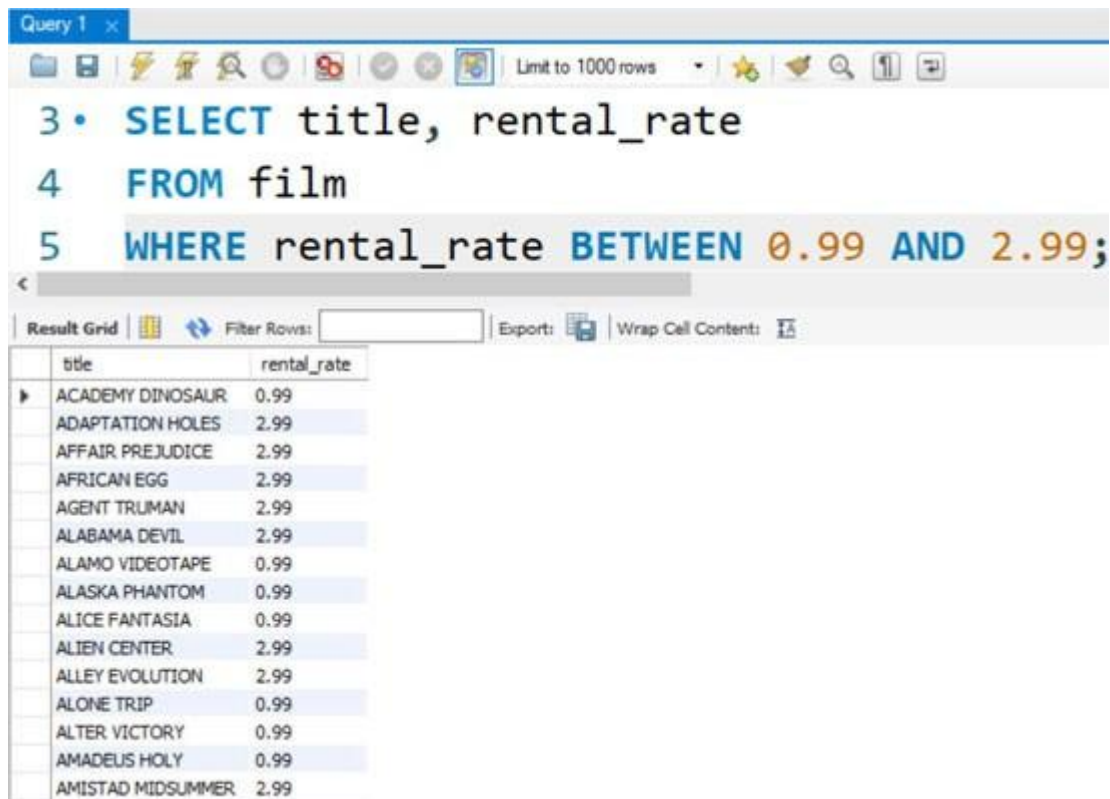


Рис. 4.2

Мы даже можем сравнить значения числового и символьного столбца с несколькими значениями, используя оператор `IN` следующим образом:

```
SELECT title, rental_rate FROM film WHERE rental_rate IN (0,
0.99, 2.99, 4.99);
SELECT title, rental_rate FROM film WHERE title IN ('ARIZONA
BANG', 'ARABIA DOGMA');
```

Оператор LIKE

Мы видели, как можно сравнивать символьные строки с помощью оператора равенства. Мы также можем сопоставлять шаблоны строк с помощью оператора `LIKE`. В дополнение к символам мы должны использовать знаки подстановки `_` и `%`. `_` означает ровно один символ, а `%` означает ноль или более символов. Давайте рассмотрим пример. Взгляните на следующий запрос:

```
SELECT * FROM film WHERE title LIKE 'A%';
```

Этот запрос вернет все строки, в которых значение столбца 'title' начинается с символа *A*. Шаблон подстановочного знака здесь означает *A*, за которым следует любое количество символов.

```
SELECT * FROM film WHERE title LIKE '_C%';
```

Этот запрос возвращает все строки, в которых значение столбца 'title' имеет второй символ как *C*. Шаблон подстановочного знака здесь означает один символ, за которым следует *C*, за которым следует любое количество символов.

```
SELECT * FROM film WHERE title LIKE '%E';
```

Этот запрос возвращает все строки, в которых значение столбца 'title' имеет последний символ как *E*. Шаблон подстановочного знака здесь означает любое количество символов и последний символ как *E*.

```
SELECT * FROM film WHERE title LIKE '%B%';
```

Этот запрос возвращает все строки, где значение заголовка столбца где угодно содержит символ *B*. Шаблон подстановочного знака здесь означает любое количество символов, за которыми следует символ *B*, за которым также следует любое количество символов.

Сравнение с NULL

Мы знаем, что NULL означает отсутствие какого-либо значения. Если мы попытаемся использовать оператор равенства для сравнения значений в столбце с NULL, то он ничего не вернет в ответ. Ниже приведен пример синтаксически правильного, но логически неправильного запроса:

```
SELECT * FROM film WHERE original_language_id = NULL;
```

Этот запрос ничего не возвращает. Ниже приведен логически правильный запрос:

```
SELECT * FROM film WHERE original_language_id IS NULL;
```

Этот запрос возвращает все строки, в которых значение столбца original_language_id равно NULL, как это показано ниже:

Query 1

Limit to 1000 rows

3

4 • **SELECT * FROM film WHERE original_language_id IS NULL;**

Result Grid

film_id	title	description	release_year	language_id	original_language_id	rental	rental_rate	length	replacement_cost	rating	sp
1	ACADEMY DINOSAUR	A Epic Drama ...	2006	1	NULL	6	0.99	86	20.99	PG	Del
2	ACE GOLDFINGER	A Astounding ...	2006	1	NULL	3	4.99	48	12.99	G	Tra
3	ADAPTATION HOLES	A Astounding ...	2006	1	NULL	7	2.99	50	18.99	NC-17	Tra
4	AFFAIR PREJUDICE	A Fanciful Doc...	2006	1	NULL	5	2.99	117	26.99	G	Cor
5	AFRICAN EGG	A Fast-Paced ...	2006	1	NULL	6	2.99	130	22.99	G	Del
6	AGENT TRUMAN	A Intrepid Pan...	2006	1	NULL	3	2.99	169	17.99	PG	Del
7	AIRPLANE SIERRA	A Touching Sa...	2006	1	NULL	6	4.99	62	28.99	PG-13	Tra
8	AIRPORT POLLOCK	A Epic Tale of ...	2006	1	NULL	6	4.99	54	15.99	R	Tra
9	ALABAMA DEVIL	A Thoughtful ...	2006	1	NULL	3	2.99	114	21.99	PG-13	Tra
10	ALADDIN CALENDAR	A Action-Pack...	2006	1	NULL	6	4.99	63	24.99	NC-17	Tra
11	ALAMO VIDEOTAPE	A Spring Epist...	2006	1	NULL	6	0.99	126	16.99	G	Cor
12	ALASKA PHANTOM	A Fanciful Sag...	2006	1	NULL	6	0.99	136	22.99	PG	Cor
13	ALI FOREVER	A Action-Pack...	2006	1	NULL	4	4.99	150	21.99	PG	Del
14	ALICE FANTASIA	A Emotional D...	2006	1	NULL	6	0.99	94	23.99	NC-17	Tra

Рис. 4.3

Логические операции

Мы также можем использовать логические операции, такие как AND (И), OR (ИЛИ) и NOT (НЕ), для объединения нескольких условий в целях фильтрации данных. Давайте рассмотрим следующие примеры:

```
SELECT * FROM film WHERE length >= 30 AND title LIKE 'A%';
```

Этот запрос следующим образом возвращает все фильмы, продолжительность которых превышает 30 минут, а название начинается с символа А:

Query 1

Limit to 1000 rows

2

3 • **SELECT * FROM film WHERE length >= 30 AND title LIKE 'A%';**

Result Grid

film_id	title	description	release_year	language_id	original_language_id	rental	rental_rate	length	replacement_cost	rating	spe
1	ACADEMY DINOSAUR	A Epic Drama ...	2006	1	NULL	6	0.99	86	20.99	PG	Delet
2	ACE GOLDFINGER	A Astounding ...	2006	1	NULL	3	4.99	48	12.99	G	Trail
3	ADAPTATION HOLES	A Astounding ...	2006	1	NULL	7	2.99	50	18.99	NC-17	Trail
4	AFFAIR PREJUDICE	A Fanciful Doc...	2006	1	NULL	5	2.99	117	26.99	G	Comm
5	AFRICAN EGG	A Fast-Paced ...	2006	1	NULL	6	2.99	130	22.99	G	Delet
6	AGENT TRUMAN	A Intrepid Pan...	2006	1	NULL	3	2.99	169	17.99	PG	Delet
7	AIRPLANE SIERRA	A Touching Sa...	2006	1	NULL	6	4.99	62	28.99	PG-13	Trail
8	AIRPORT POLLOCK	A Epic Tale of ...	2006	1	NULL	6	4.99	54	15.99	R	Trail
9	ALABAMA DEVIL	A Thoughtful ...	2006	1	NULL	3	2.99	114	21.99	PG-13	Trail
10	ALADDIN CALENDAR	A Action-Pack...	2006	1	NULL	6	4.99	63	24.99	NC-17	Trail
11	ALAMO VIDEOTAPE	A Spring Epist...	2006	1	NULL	6	0.99	126	16.99	G	Comm
12	ALASKA PHANTOM	A Fanciful Sag...	2006	1	NULL	6	0.99	136	22.99	PG	Comm
13	ALI FOREVER	A Action-Pack...	2006	1	NULL	4	4.99	150	21.99	PG	Delet
14	ALICE FANTASIA	A Emotional D...	2006	1	NULL	6	0.99	94	23.99	NC-17	Trail
15	ALIEN CENTER	A Brilliant Dra...	2006	1	NULL	5	2.99	46	10.99	NC-17	Trail

Рис. 4.4

Ниже приведен пример операции OR:

```
SELECT * FROM film WHERE length >= 30 OR title LIKE 'A%';
```

Аналогично, ниже приведен пример операции NOT:

```
SELECT * FROM film WHERE rental_duration NOT IN (1, 2, 3, 5, 6);
```

Давайте посмотрим, как можно написать запрос типа BETWEEN AND с логическими операциями. Ниже приведен запрос:

```
SELECT title, rental_rate FROM film WHERE rental_rate BETWEEN 1.99 AND 2.99;
```

Мы можем переписать это с помощью логических операций следующим образом:

```
SELECT title, rental_rate FROM film WHERE rental_rate >= 0.99 AND rental_rate <= 2.99;
```

Мы также можем переписать запросы, использующие оператор IN, с помощью логического оператора. Ниже приведен запрос с оператором IN:

```
SELECT title, rental_rate FROM film WHERE title IN ('ARIZONA BANG', 'ARABIA DOGMA');
```

Ниже приведен переписанный запрос с логическим оператором OR:

```
SELECT title, rental_rate FROM film WHERE title = 'ARIZONA BANG' OR title = 'ARABIA DOGMA';
```

Сортировка набора результатов

Мы можем с помощью оператора ORDER BY следующим образом отсортировать данные, полученные запросом SELECT:

```
SELECT * FROM film ORDER BY title;
```

Запрос возвращает набор результатов, отсортированный в алфавитном порядке по возрастанию заголовка символьного столбца.

По умолчанию все данные, возвращаемые с помощью оператора ORDER

BY, располагаются по возрастанию в алфавитном порядке в случае символьных столбцов и в порядке следования чисел в случае числовых столбцов. Мы можем получить их в порядке возрастания или убывания, явно используя ключевые слова ASC и DESC.

```
SELECT * FROM film ORDER BY title ASC;
SELECT * FROM film ORDER BY title DESC;
```

Мы также можем следующим образом использовать для сортировки псевдонимы столбцов:

```
SELECT title AS NAME, description FROM film ORDER BY NAME;
SELECT title AS NAME, description FROM film ORDER BY NAME
DESC;
```

Мы также можем сортировать по нескольким столбцам следующим образом:

```
SELECT rental_rate, title, description FROM film ORDER BY
rental_rate, title;
SELECT rental_rate, title, description FROM film ORDER BY
rental_rate, title DESC;
```

Обратите внимание, что при этом сортируются только полученные данные, а не данные, хранящиеся в таблицах базы данных.

Работа с датами

Мы увидели, как использовать различные операторы с числовыми и символьными типами столбцов. Теперь мы увидим, как работать со столбцами типов datetime. Мы подробно рассмотрим типы данных в MySQL, изучая запросы языка определения данных **Data Definition Language (DDL)**.

Следующий запрос фильтрует выходные данные на основе точной даты и времени:

```
SELECT * FROM rental WHERE rental_date = '2005-05-24
22:53:30';
```

Ниже приведен пример столбца даты и времени с оператором сравнения:

```
SELECT * FROM rental WHERE rental_date > '2005-05-24';
```

Мы также можем использовать BETWEEN AND для извлечения данных в диапазоне дат следующим образом:

```
SELECT * FROM rental WHERE rental_date BETWEEN '2005-05-24'  
AND '2005-05-29';
```

Мы также можем использовать оператор LIKE для извлечения всех записей 2005 года следующим образом:

```
SELECT * FROM rental WHERE rental_date LIKE '2005%';
```

Заключение

В этой главе мы изучили и продемонстрировали подробное использование оператора WHERE для фильтрации набора результатов, возвращаемого запросом SELECT. Теперь мы можем писать различные запросы SELECT с оператором WHERE для получения данных в соответствии с потребностями последующих бизнес- или научных приложений.

Оператор WHERE — важный аспект SQL и одна из наиболее часто используемых функций SQL.

В следующей главе мы начнем с концепции встроенных функций MySQL и подробно изучим однострочные функции.

Что следует помнить

- NULL нельзя сравнивать ни с каким значением. Мы должны использовать IS NULL и IS NOT NULL, чтобы использовать его в операторе WHERE.
- Мы можем работать с операторами сравнения и столбцами datetime.
- Мы можем использовать оператор ORDER BY для сортировки resultset по возрастанию и убыванию столбцов datetime, numerical и character.

Вопросы с выбором верного ответа

1. Какое предложение фильтрует набор результатов?

a) GROUP BY

- b)* FILTER
- c)* WHERE
- d)* ORDER BY

2. Какой оператор используется для сравнения строковых шаблонов с символьными столбцами?

- a)* COMPARE
- b)* LIKE
- c)* =
- d)* REGEX

3. Какой оператор используется для сортировки набора результатов?

- a)* FILTER
- b)* SORT
- c)* WHERE
- d)* ORDER BY

Ответы на Вопросы

- 1. *c)*
- 2. *b)*
- 3. *d)*

Вопросы

- 1. Как отсортировать набор результатов, возвращаемый запросом?
- 2. Как сравнить NULL?
- 3. Что такое логические операторы?
- 4. Как сопоставить шаблоны строк в символьных столбцах?

Ключевые термины

WHERE, LIKE, ORDER BY, IS NULL

ГЛАВА 5

Однострочные функции

В предыдущей главе мы изучили и продемонстрировали оператор `WHERE`. Мы научились фильтровать строки выходного набора результатов с помощью оператора `WHERE`.

В этой главе мы сосредоточимся на основах функций MySQL и MariaDB. Мы изучим и подробно продемонстрируем функции одной строки. Мы рассмотрим различные категории однострочных функций, такие как манипуляции с регистром и символами, манипуляции с числами и манипуляции с датами. Мы также научимся обрабатывать с помощью этих функций значения `NULL`.

Структура

Ниже приведен подробный список тем, которые мы изучим и продемонстрируем с помощью SQL-запросов в этой главе:

- Функции одной строки
- Двойная таблица
- Функции манипуляции с регистрами
- Функции манипуляции с символами
- Функции манипуляции с числами
- Функции манипуляции с датами
- Datetime в строку и наоборот
- Обработка `NULL`
- Оператор `CASE`
- Вложенные однострочные функции

Цель

Цель этой главы — помочь читателям освоить функции SQL. Мы подробно изучим и продемонстрируем все типы однострочных функций.

Прочитав эту главу, читатели смогут работать с однострочными функциями различных типов и применять их к набору результатов, возвращаемому запросом `SELECT`.

Однострочные функции

MySQL имеет множество встроенных функций для выполнения различных операций с данными. Функции — это подпрограммы, которые выполняют вычисления с данными, которые мы передаем им в качестве аргументов. Для баз данных данные могут быть постоянными данными или столбцами. Эти функции полезны при выполнении таких операций, как математические вычисления, конкатенация строк, подстроки и т. д. SQL-функции делятся на два типа:

- Однострочные функции (также известные как скалярные функции)
- Агрегатные функции (также известные как групповые функции)

В этой главе основное внимание будет уделено однострочным функциям. Однострочные функции принимают n строк в качестве входных данных и возвращают такое же количество n строк в выходном наборе результатов. Они применяются к каждой строке с целью получения результатов. Существуют различные типы однострочных функций, и мы изучим и продемонстрируем эти категории одну за другой.

Двойная таблица

Давайте начнем с понимания концепции таблицы `DUAL`. Это фиктивная таблица, состоящая из одной строки и одного столбца. Чаще всего она используется в качестве заполнителя в SQL-запросах, которым не нужна реальная таблица. Ниже приведены примеры запросов с использованием такой таблицы:

```
USE Sakila;
SELECT 1 from DUAL;
SELECT 1 + 1 from DUAL;
SELECT 'Oracle' from DUAL;
```

Первый и второй запрос выбирают числа, а третий запрос выбирает строку.

ПРИМЕЧАНИЕ: В MySQL и MariaDB нам не требуется таблица `DUAL`, и те же запросы можно выполнять без опции `из DUAL`. Т.е. `SELECT 1 from DUAL`

и `SELECT 1` будут работать одинаково в MySQL и MariaDB. В Oracle таблица требуется всегда, отсюда и концепция фиктивной таблицы.

Функции управления регистром

Функции управления регистром используются для управления регистром символов в символьных строках. Мы можем применять их к статическим строкам или символьным столбцам. Давайте продемонстрируем это. Если мы запустим следующий запрос:

```
SELECT title, description FROM film;
```

Результат будет следующим:

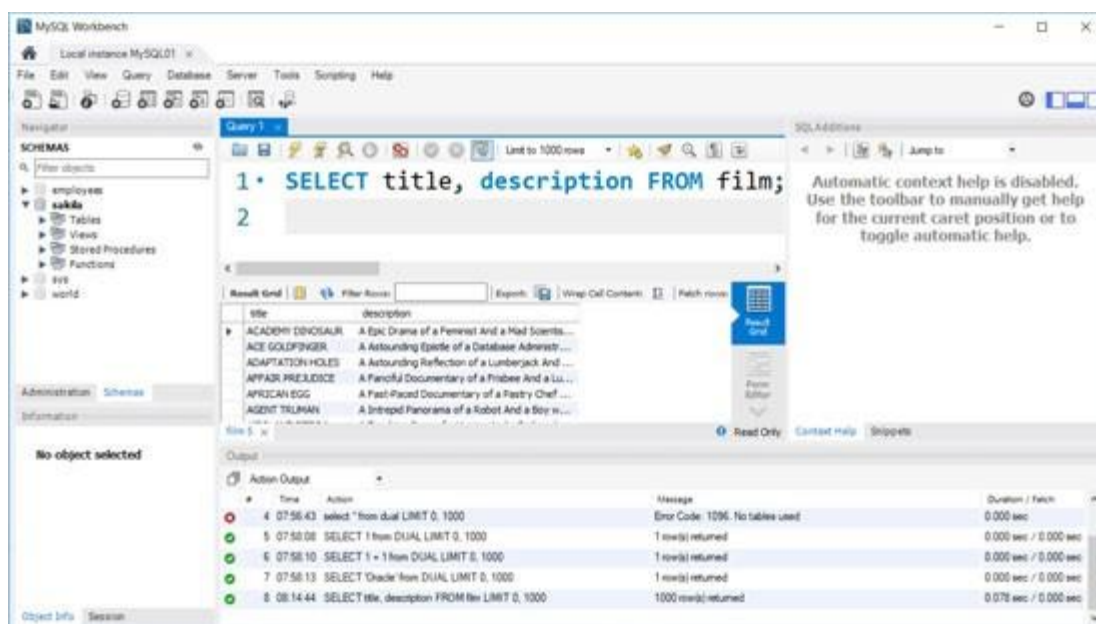


Рис. 5.1

Мы видим, что все символы в первом столбце вывода имеют прописные буквы, а во втором столбце — смешанный регистр. Мы можем изменить это, выполнив следующий запрос:

```
SELECT title, LOWER(title) FROM film;
```

В выводе окажется заголовок в первом столбце, а все символы в нижнем регистре во втором столбце вывода, как это показано ниже:

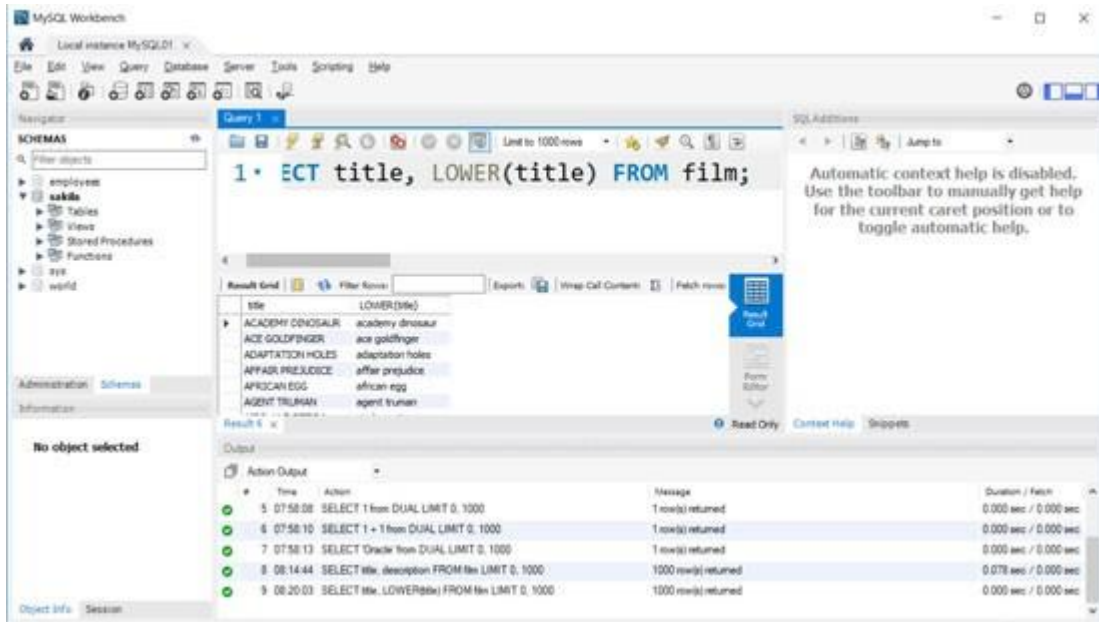


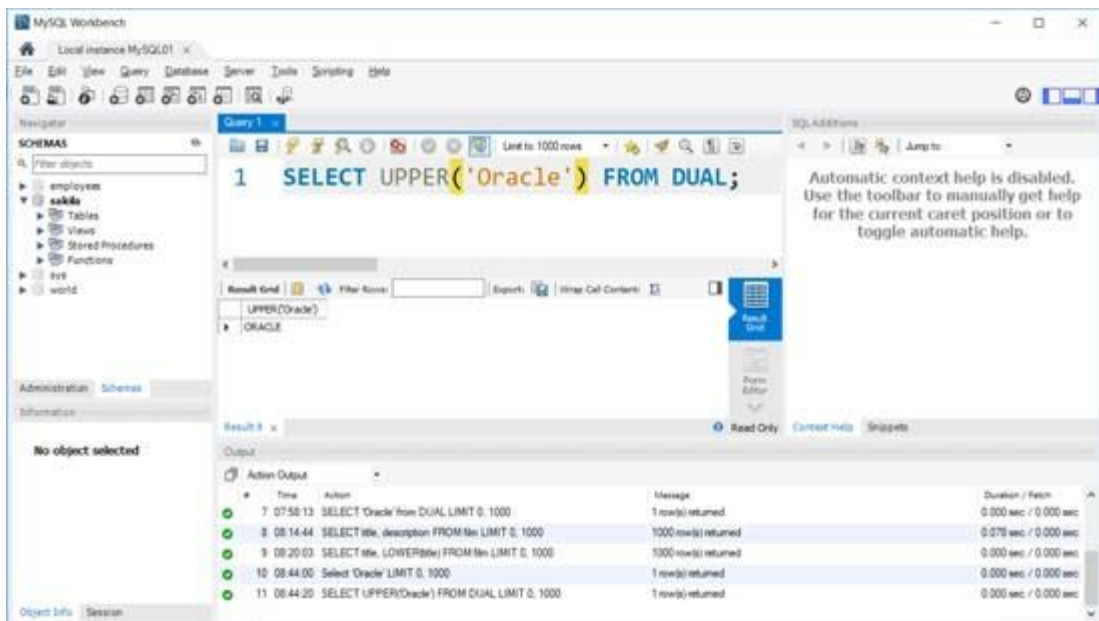
Рис. 5.2

Давайте продемонстрируем еще одну функцию:

`SELECT description, Upper(description) FROM film;`

Запустите этот запрос; вывод будет содержать столбец описания и верхний регистр этого столбца.

Мы можем также использовать его со строками следующим образом:



Функции управления символами

Функции манипуляции символами можно использовать для управления строками символов. Таких функций много, и в этом разделе мы будем демонстрировать их одну за другой. Мы можем использовать с этими функциями строки или символьные столбцы.

Мы можем объединить две строки с помощью функции `CONCAT()`. Она работает так же, как оператор `||`. Ниже приведены примеры того же самого запроса:

```
SELECT CONCAT('Oracle', ' SQL') FROM DUAL;  
SELECT CONCAT(title, description) FROM film;
```

В первом примере мы объединяем две строки, а во втором — два разных символьных столбца. Это одна из простейших функций манипулирования символами.

Мы можем извлечь подстроку из строки с помощью функции `SUBSTR()` следующим образом:

```
SELECT SUBSTR('Oracle SQL', 1, 6) FROM DUAL;
```

Индексация строк начинается с 1. Первый аргумент — это строка или столбец, из которого нам нужно извлечь подстроку. Второй и третий аргументы — это начальный и конечный индексы подстроки. Приведенный выше запрос выведет строку 'Oracle'.

Мы можем использовать функцию `length()` для определения длины строки или символьного столбца следующим образом:

```
SELECT title, LENGTH(title) FROM film;
```

Мы можем узнать расположение подстроки внутри строки, используя функцию `INSTR()` следующим образом:

```
SELECT INSTR('Oracle SQL', 'S') FROM DUAL;  
SELECT INSTR('Oracle SQL', 'l') FROM DUAL;
```

В обоих примерах мы используем функцию для определения местоположения подстроки (которая передается в качестве второго аргумента).

Мы можем использовать функции `LPAD()` и `RPAD()` для дополнения строки определенным символом. Запустите следующий запрос, чтобы увидеть `LPAD()` в действии:

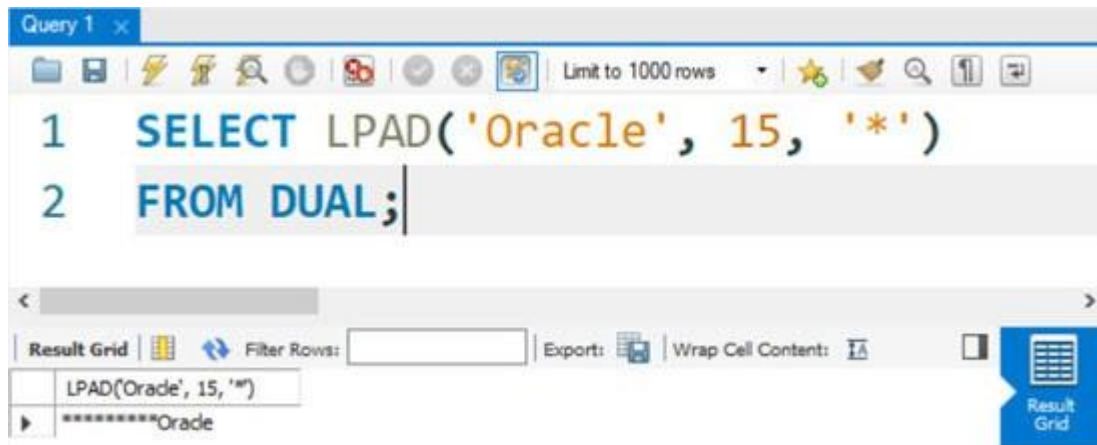


Рис. 5.4

Ниже приведен пример `RPAD()` в действии:

```
SELECT RPAD('Oracle', 15, '*') FROM DUAL;
```

Функции манипулирования числами

Мы можем округлять и усекать числа. Для этого в MySQL есть функции `ROUND()` и `TRUNCATE()`. Они принимают число и позицию числа в качестве аргументов. Позиция отрицательного целого числа означает, что позиция находится перед десятичной запятой, 0 означает десятичную точку, а положительное целое число означает, что это позиция после десятичной точки. Функция `ROUND()` округляет число, а функция `TRUNCATE()` усекает число. Ниже приведены примеры запросов:

```
SELECT ROUND(67.1234, 1), ROUND(67.1234, 0), ROUND(67.1234, -1) FROM DUAL;
```

```
SELECT ROUND(67.1234, 2), ROUND(67.1234, 0), ROUND(67.1234, -2) FROM DUAL;
```

```
SELECT TRUNCATE(67.1234, 1) from dual;
```

```
SELECT TRUNCATE(67.1234, 0) from dual;
```

```
SELECT TRUNCATE(67.1234, -1) from dual;
```

Запустите запросы, приведенные выше, и посмотрите результат.

Мы также можем использовать функцию `MOD()` для вычисления модуля:

```
SELECT MOD(15, 6) from dual;
```

Запустите запрос, приведенный выше, и посмотрите результат.

Функции манипулирования датами

Мы можем получить текущую дату следующим образом:

```
SELECT sysdate() from DUAL;
```

Мы можем вычислить разницу между двумя датами следующим образом:

```
SELECT TIMESTAMPDIFF(DAY, '2012-05-05', sysdate()) FROM DUAL;  
SELECT TIMESTAMPDIFF(MONTH, '2012-05-05', sysdate()) FROM  
DUAL;
```

Первый аргумент — это единица измерения разницы, а остальные два аргумента — это даты. Это могут быть столбцы дат или абсолютные даты, как это показано выше. Наконец, мы можем добавить несколько дней к дате следующим образом:

```
SELECT DATE_ADD(sysdate(), INTERVAL 10 DAY);  
SELECT DATE_ADD(sysdate(), INTERVAL -10 DAY);
```

Вышеупомянутые вызовы функций возвращают дату. Обратите внимание, что мы можем изменить единицу измерения времени на MONTH (МЕСЯЦ), YEAR (ГОД) или любую другую желаемую единицу в соответствии с нашими требованиями. Отрицательное значение означает вычитание.

Дата и время в строку и наоборот

Мы можем преобразовать дату и время в строку и наоборот. Мы можем использовать функцию STR_TO_DATE() для преобразования любой форматированной строки в дату. Первый аргумент — это строка, а второй аргумент — это формат, используемый для представления даты в строке. Ниже приведены примеры вызовов функций:

```
SELECT STR_TO_DATE('25/08/1986', '%d/%m/%Y') FROM DUAL;  
SELECT STR_TO_DATE('25-08-1986', '%d-%m-%Y') FROM DUAL;  
SELECT STR_TO_DATE('15-AUG-1947', '%d-%M-%Y') FROM DUAL;
```

Аналогичным образом мы можем использовать функцию

DATE_FORMAT() для преобразования даты в строку в желаемом формате следующим образом:

```
SELECT DATE_FORMAT(SYSDATE(), '%Y-%m-%d') FROM DUAL;  
SELECT DATE_FORMAT(SYSDATE(), '%Y-%m-%d %H:%i:%s') FROM DUAL;  
SELECT DATE_FORMAT(SYSDATE(), '%d-%b-%Y') FROM DUAL;
```

Первый аргумент — это дата, которую нужно преобразовать в строку, а второй аргумент — это формат, в котором строка должна отображаться.

Обработка NULL

Давайте посмотрим на несколько функций, которые обрабатывают значение NULL. Если значение равно NULL, мы можем заменить его функцией IFNULL(). Ниже приведен пример:

```
select original_language_id, IFNULL (original_language_id,  
'English') from film;
```

В приведенном выше запросе, если значение первого аргумента равно NULL, оно заменяется вторым аргументом.

Функция NULLIF() возвращает NULL, если два выражения равны, иначе возвращается первый переданный аргумент. Мы можем использовать ее следующим образом:

```
Select first_name, last_name, length(first_name),  
length(last_name),  
NULLIF(length(first_name), length(last_name)) from actor;
```

В приведенном выше примере мы сравниваем длину двух разных символьных столбцов. Результат выглядит следующим образом:

Query 1

```

1 • Select first_name, last_name,
2 length(first_name), length(last_name),
3 NULLIF(length(first_name), length(last_name)) from actor;

```

first_name	last_name	length(first_name)	length(last_name)	NULLIF(length(first_name), length(last_name))
PENELOPE	GUINNESS	8	7	8
NICK	WAHLBERG	4	8	4
ED	CHASE	2	5	2
JENNIFER	DAVIS	8	5	8
JOHNNY	LOLLOBRIGIDA	6	12	6
BETTE	NICHOLSON	5	9	5
GRACE	MOSTEL	5	6	5
MATTHEW	JOHANSSON	7	9	7
JOE	SWANK	3	5	3
CHRISTIAN	GABLE	9	5	9
ZERO	CAGE	4	4	NULL
KARL	BERRY	4	5	4
LMA	WOOD	3	4	3
VIVIAN	BERGEN	6	6	NULL

Рис. 5.5

Если мы хотим выбрать первое выражение, отличное от NULL, из нескольких выражений, нам нужно использовать функцию `coalesce()`:

```
select coalesce(address2, address, district) from address;
```

Оператор CASE

Мы можем иметь несколько блоков кода с помощью оператора `CASE`. Это позволяет нам следующим образом обрабатывать несколько возможностей для выражения:

```

SELECT film_id, title, description, release_year,
rental_rate,
CASE rental_rate
WHEN 0.99 THEN 1.00
WHEN 1.99 THEN 2.5
WHEN 2.99 THEN 4.5
ELSE rental_rate END AS "Revised Rental"
FROM film;

```

В приведенном выше запросе мы можем обрабатывать несколько случаев для значения столбца. Результат следующий:

The screenshot shows a SQL query editor window titled "Query 1". The query is as follows:

```

1 • SELECT film_id, title, description,
2   release_year, rental_rate,
3   CASE rental_rate
4   WHEN 0.99 THEN 1.00
5   WHEN 1.99 THEN 2.5
6   WHEN 2.99 THEN 4.5
7   ELSE rental_rate END AS "Revised Rental"
8   FROM film;

```

Below the query, the "Result Grid" is displayed with the following data:

film_id	title	description	release_year	rental_rate	Revised Rental
1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientis...	2006	0.99	1.00
2	ACE GOLDFINGER	A Astounding Epistle of a Database Administr...	2006	4.99	4.99
3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And ...	2006	2.99	4.5
4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lu...	2006	2.99	4.5
5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef ...	2006	2.99	4.5
6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy w...	2006	2.99	4.5

Рис. 5.6

Вложенные однострочные функции

Мы можем использовать функции одной строки вложенным способом. В этом методе мы передаем вызов функции вызову другой функции следующим образом:

```
Select LOWER(SUBSTR(title, 1, 5)) from film;
```

В приведенном выше запросе мы вызываем функцию SUBSTR() внутри функции LOWER(). Запустите запрос и посмотрите результат сами.

Заключение

В этой главе мы изучили и продемонстрировали подробное использование функций одной строки в MySQL. Мы видели почти все категории однострочных функций. Эти однострочные функции очень удобны при операциях, связанных с преобразованием данных и созданием отчетов. Часто

нам может потребоваться правильно отформатировать и обработать данные или получить новые данные на основе существующих данных в базе данных. Эти функции удовлетворяют эту потребность. В следующей главе мы рассмотрим агрегатные функции и обсудим их подробно.

Что следует помнить

- Однострочные функции принимают набор результатов с n числом строк и возвращают n количество строк.
- Функции одной строки используются для получения новых данных, преобразования данных или создания отчетов.
- Мы можем использовать несколько функций одной строки в одном запросе SQL. Мы также можем использовать их вложенным образом.

Вопросы с выбором правильного ответа

1. Что является синонимом термина **однострочная функция**?
 - a) Скалярная функция
 - b) Векторная функция
 - c) Групповая функция
 - d) Агрегатная функция
2. Для какой задачи НЕ следует использовать однострочные функции?
 - a) Манипулирование символами
 - b) Манипулирование числами
 - c) Группировка и агрегирование данных
 - d) Обработка NULL
3. Что является синонимом термина **групповая функция**?
 - a) Скалярная функция
 - b) Векторная функция
 - c) Однострочная функция
 - d) Агрегатная функция

Ответы на Вопросы

1. *a)*
2. *c)*
3. *d)*

Вопросы

1. Что такое однострочные функции?
2. Как использовать ключевое слово `CASE`?
3. Объясните сценарии, в которых мы можем использовать однострочные функции.
4. Как обрабатывать `NULL`-значения выражения?
5. Как добавить год к текущей дате? Реализуйте это с помощью SQL-запроса.

Ключевые термины

Однострочные функции, вложенные однострочные функции, манипуляция с символами, `CASE`, манипуляция с регистром, численные функции

ГЛАВА 6

Групповые функции

В последней главе мы изучили и продемонстрировали однострочные функции (также известные как **скалярные функции**). Мы увидели различные категории однострочных функций и научились их использовать. Мы также обсудили сценарии, в которых мы могли бы использовать однострочные функции. В этой главе мы продолжим наше стремление более подробно изучить функции MySQL и MariaDB.

В этой главе мы изучим и продемонстрируем групповые функции. Групповые функции — это полезные функции любой СУБД. Они часто используются при анализе данных, где требуется группировка данных. Оператор `GROUP BY` используется для группировки данных. Мы также изучим оператор `HAVING` и продемонстрируем его при фильтрации групп. Мы изучим и продемонстрируем все эти темы подробно в этой главе.

Структура

Ниже приведен подробный список тем, которые мы изучим и продемонстрируем с помощью SQL-запросов в этой главе:

- Групповые функции
- Оператор `GROUP BY`
- `HAVING`

Цель

Мы подробно изучим и продемонстрируем все групповые функции. Прочитав эту главу, читатели смогут работать с групповыми функциями и использовать оператор `HAVING`.

Групповые функции

Групповые функции также известны как агрегатные функции. Эти

функции работают с наборами строк и возвращают один результат для каждого набора. Наборы строк определяются по некоторым критериям. Давайте изучим, как использовать функции для группировки. У нас есть следующие групповые функции в MySQL:

- ♦ **AVG ()** : вычисляет среднее значение
- ♦ **MAX ()** : возвращает максимум набора
- ♦ **MIN ()** : возвращает минимум набора
- ♦ **SUM ()** : возвращает сумму набора
- ♦ **STDDEV ()** : вычисляет стандартное отклонение
- ♦ **VARIANCE ()** : вычисляет дисперсию

Следующие запросы демонстрируют использование этих функций:

```
USE
sakila;
SELECT AVG(address_id) FROM staff;
SELECT MAX(address_id) FROM staff;
SELECT MIN(address_id) FROM staff;
SELECT SUM(address_id) FROM staff;
SELECT STDDEV(address_id) FROM staff;
SELECT VARIANCE(address_id) FROM staff;
```

Все приведенные выше запросы рассматривают всю таблицу как один набор, и мы получаем для каждого запроса в качестве результата только одну строку.

Все групповые функции по умолчанию игнорируют значения NULL. Например, запрос `SELECT AVG(postal_code) FROM address` игнорирует нулевые значения. Среднее значение рассчитывается только для строк, в которых значение столбца почтового индекса не равно NULL.

Чтобы обойти это, необходимо использовать следующий запрос:

```
SELECT AVG(NULLIF(postal_code, 0)) FROM address;
```

Приведенный выше запрос заменяет значения NULL нулями; таким образом, функция `AVG ()` также будет учитывать строки со значениями NULL при окончательном вычислении.

Другая специальная групповая функция — `COUNT ()`. Она возвращает количество строк. Если мы запустим запросы `SELECT COUNT (*) FROM`

address или `SELECT COUNT(1) FROM address`, то она вернет подсчет общего количества строк независимо от значений `NULL`. Если мы запустим такой запрос, как `SELECT count(address2) FROM address`, она вернет количество строк, не содержащих `NULL`, для этого столбца.

Мы даже можем узнать количество различных значений для столбца следующим образом:

```
SELECT COUNT( DISTINCT postal_code) FROM address;
```

Вот как мы можем использовать групповые функции для полной таблицы.

Оператор GROUP BY

До сих пор мы видели, как использовать групповые функции для всей таблицы. Мы можем разделить набор результатов на отдельные наборы, чтобы получить одно значение для каждого набора, используя предложение `GROUP BY`. Например, мы можем использовать его следующим образом:

```
SELECT SUM(rental_rate)
FROM film
GROUP BY rental_duration ;
```

В приведенном выше запросе мы вычисляем сумму ставок аренды, сгруппированных по продолжительности аренды. Чтобы представить это в лучшем контексте, мы можем использовать следующий запрос:

```
SELECT rental_duration, SUM(rental_rate)
FROM film
GROUP BY rental_duration ;
```

Результат будет следующий:

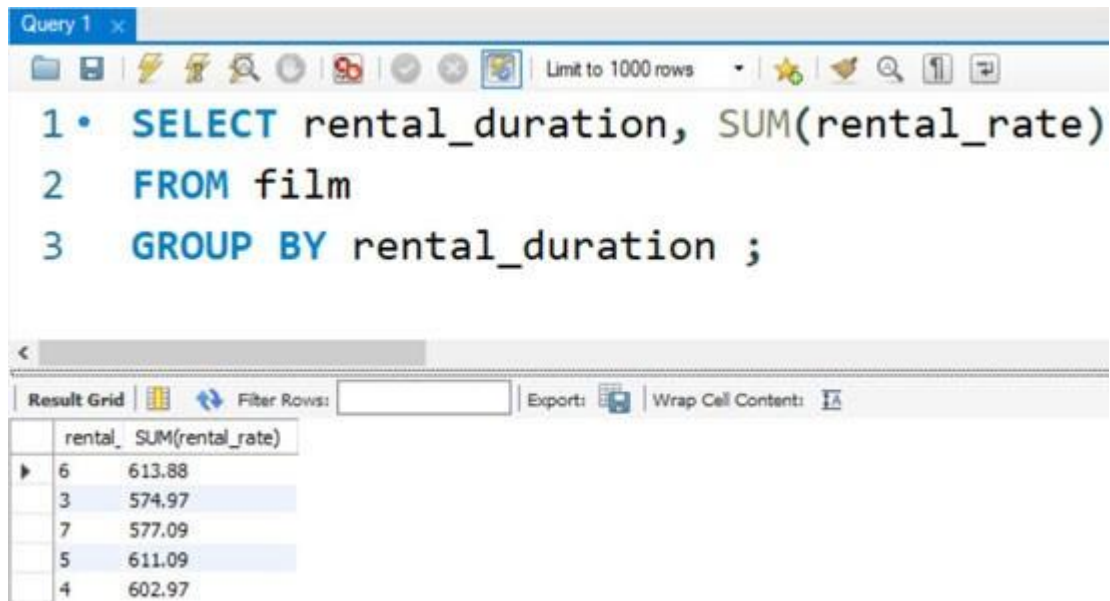


Рис. 6.1

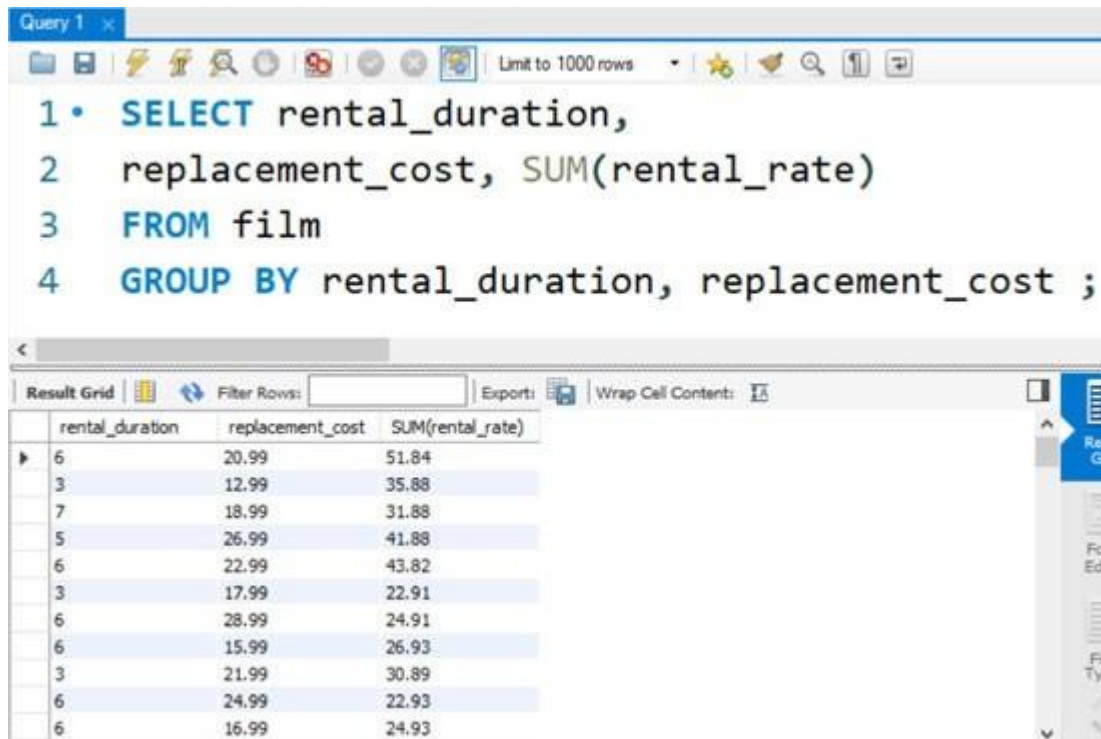
Мы также можем использовать другие групповые функции таким же образом, как это показано ниже:

```
SELECT rental_duration, AVG(rental_rate)
FROM film
GROUP BY rental_duration;
```

Мы также можем использовать больше столбцов в операторе `GROUP BY`. Единственное условие заключается в том, что все эти столбцы также должны находиться в списке столбцов `SELECT`, как это показано ниже:

```
SELECT rental_duration, replacement_cost, SUM(rental_rate)
FROM film
GROUP BY rental_duration, replacement_cost ;
```

Для нескольких столбцов в операторе `GROUP BY` результат будет следующим :



```

1 • SELECT rental_duration,
2     replacement_cost, SUM(rental_rate)
3     FROM film
4     GROUP BY rental_duration, replacement_cost ;

```

rental_duration	replacement_cost	SUM(rental_rate)
6	20.99	51.84
3	12.99	35.88
7	18.99	31.88
5	26.99	41.88
6	22.99	43.82
3	17.99	22.91
6	28.99	24.91
6	15.99	26.93
3	21.99	30.89
6	24.99	22.93
6	16.99	24.93

Рис. 6.2

HAVING

Мы уже видели, как фильтровать строки на основе некоторых условий с помощью оператора `WHERE`. Ограничением оператора `WHERE` является то, что он фильтрует только строки, но не группы. Чтобы фильтровать группы, нам нужно использовать оператор `HAVING`. Ниже приведен пример запроса, который демонстрирует это:

```

SELECT rental_duration, SUM(rental_rate)
FROM film
GROUP BY rental_duration
HAVING SUM(rental_rate) > 600;

```

Мы фильтруем группы выходных данных на основе критериев, упомянутых в операторе `HAVING`. Не обязательно использовать одну и ту же агрегатную функцию в операторе `HAVING`. Мы также можем использовать разные функции. Следующий запрос демонстрирует это:

```

SELECT rental_duration, SUM(rental_rate)
FROM film
GROUP BY rental_duration

```



```
HAVING AVG(rental_rate) > 3;
```

Заключение

В этой главе мы изучили и продемонстрировали подробное использование функций `group by` (также известных как **агрегатные**) MySQL. Мы рассмотрели все агрегатные функции. Групповые функции, как и однострочные функции, также используются для формирования отчетов и преобразования данных. Они полезны при группировке данных. Мы можем фильтровать сгенерированные ими данные с помощью оператора `HAVING`.

Что следует помнить

- Агрегатные функции также известны как групповые функции
- Агрегатные функции также используются для получения новых данных, преобразования данных или создания отчетов
- Мы можем использовать несколько групповых функций в одном SQL-запросе
- Мы должны использовать предложение `HAVING` для фильтрации групп

Вопросы с выбором верного ответа

1. Какой оператор фильтрует сгруппированные данные?
 - a) `HAVING`
 - b) `WHERE`
 - c) `GROUP BY`
 - d) `ORDER BY`
2. Для какой задачи лучше всего использовать агрегатные функции?
 - a) Манипулирование символами
 - b) Манипулирование числами
 - c) Группировка и агрегирование данных
 - d) Обработка `NULL`

Ответы на Вопросы

1. a)
2. c)

Вопросы

1. Что такое агрегатные функции? Чем они отличаются от однострочных функций?
2. Объясните сценарии, в которых мы можем использовать агрегатные функции.
3. Как агрегатные функции обрабатывают значения NULL?

Ключевые термины

Агрегатные функции, групповые функции, агрегирование данных, группировка, HAVING, GROUP BY.

ГЛАВА 7

Объединения в MySQL

В предыдущей главе мы изучили и продемонстрировали групповые функции (также известные как агрегатные функции). Мы также обсудили сценарии, в которых мы могли бы использовать групповые функции.

В этой главе мы изучим и продемонстрируем методы объединения столбцов из разных источников. Эти методы известны как объединение. Они очень полезны во многих задачах бизнес-аналитики, когда нам нужны данные из нескольких источников. Людям, работающим в проектах, связанных с анализом и визуализацией данных, приходится часто использовать объединение. Если вы хотите работать администратором базы данных, аналитиком данных или специалистом по данным, то знание объединения является обязательным.

Структура

Ниже приведен подробный список тем, которые мы изучим и продемонстрируем с помощью SQL-запросов в этой главе:

- Концепция объединений
- Перекрестное объединение
- Простое/естественное/внутреннее объединение
- Внешнее объединение
- Несколько таблиц и условий при объединении
- Самообъединение
- Неравноправное объединение

Цель

Мы подробно изучим и продемонстрируем все типы объединений. Прочитав эту главу, читатели смогут успешно комбинировать столбцы из различных источников для получения данных в соответствии с

потребностями своего бизнеса и аналитики данных.

Концепция объединений

Объединения являются важной особенностью любой СУБД. Они часто используются в запросах, связанных с анализом данных. Объединения используются для объединения данных из нескольких источников (таблиц или представлений). Мы объединяем данные из разных источников в соответствии с каким-то условием. В зависимости от того, какие условия мы используем при объединениях, они подразделяются на следующие типы:

- Эквивалентные объединения
- Неэквивалентные объединения

Эквивалентные объединения используют оператор равенства (=) в условии объединения. В неэквивалентных объединениях используются другие операторы, такие как `LIKE` и `>=`, в качестве условия объединения. Мы подробно рассмотрим как типы соединений, так и подтипы эквивалентных объединений. Ранее мы узнали, что объединения объединяют столбцы из нескольких источников. Источниками могут быть таблицы и представления или их комбинация; то есть один из источников может быть таблицей, а другой — представлением. До сих пор мы использовали таблицы в наших запросах `SELECT`. Мы изучим и продемонстрируем представления в 10-й главе, подробно посвященной понятию представлений.

Перекрестное объединение

Перекрестное объединение также известно как **декартово произведение**. Это набор всех комбинаций всех строк из всех источников, упомянутых в запросе. Если запрос имеет две таблицы в качестве источника с количеством строк n и m соответственно, то набор результатов перекрестного объединения будет иметь количество строк $n \times m$.

Есть два способа указать объединение. Старый синтаксис не указывает его в запросе. Стандарт ANSI позволяет нам указать его в самом запросе. Для каждого типа объединения мы рассмотрим оба синтаксиса.

Мы можем перекрестно объединить две таблицы, используя следующий

синтаксис:

```
Use  
sakila;  
SELECT * FROM country, city;
```

Результат будет следующий:

country_id	country	last_update	city_id	city	country_id	last_update
1	Afghanistan	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
2	Algeria	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
3	American Samoa	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
4	Angola	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
5	Anguilla	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
6	Argentina	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
7	Armenia	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
8	Australia	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
9	Austria	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
10	Azerbaijan	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
11	Bahrain	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
12	Bangladesh	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
13	Belarus	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
14	Bolivia	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
15	Brazil	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
16	Brunei	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
17	Bulgaria	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
18	Cambodia	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
19	Cameroon	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
20	Canada	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
21	Chad	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
22	Chile	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25
23	China	2006-02-15 04:44:00	1	A Corua (La Corua)	87	2006-02-15 04:45:25

Рис. 7.1

Синтаксис ANSI для него следующий:

```
SELECT * FROM country CROSS JOIN city;
```

Часто говорят, что декартово произведение или перекрестное объединение — это не что иное, как обычное объединение без каких-либо условий. Таким образом, мы можем получить тот же результат, выполнив следующий запрос:

```
SELECT * FROM country JOIN city;
```

Простые/Внутренние и естественные объединения

Другой тип объединения, при котором данные из нескольких таблиц объединяются на основе соответствия условию, известен как **простое** или **внутреннее объединение**. Старый синтаксис или синтаксис тета-стиля для простого/внутреннего объединения:

```
select * from city, country where city.country_id =  
country.country_id;
```

В этом запросе мы сопоставляем данные таблиц городов и стран на основе равенства значений общего столбца `country_id`, присутствующего в обеих таблицах. Все записи обеих таблиц объединяются, при этом значения столбцов, упомянутых в условии, совпадают. Столбцы, используемые в условии объединения, не обязательно должны иметь одинаковые наименования, но столбцы должны иметь одинаковый тип данных и размер. В приведенном выше запросе мы используем наименования таблиц для обращения к столбцам, поскольку наименования столбцов одинаковы в обеих таблицах. Мы можем написать тот же запрос, используя псевдонимы таблиц следующим образом:

```
select * from city ci, country co where ci.country_id =  
co.country_id;
```

Это был синтаксис старого стиля. Синтаксис ANSI-стиля будет выглядеть следующим образом:

```
SELECT city, country FROM city INNER JOIN country ON  
city.country_id = country.country_id;
```

Другой синтаксис ANSI-стиля для того же запроса выглядит следующим образом:

```
SELECT city, country FROM city INNER JOIN country USING  
(country_id);
```

Чтобы понять концепцию естественного объединения, нам нужно переключиться на другую базу данных, используя следующий запрос:

```
USE
```

```
employees;
```

Естественное объединение — это объединение столбцов, имеющих одинаковые наименования, типы данных и размеры в разных источниках. Ниже показано естественное объединение двух таблиц:

```
SELECT * FROM employees NATURAL JOIN dept_emp;
```

Если мы опишем обе таблицы с помощью запросов `desc employees` и `desc dept_emp`, мы обнаружим, что столбцы `emp_no` совпадают в обеих таблицах. Приведенный выше запрос эквивалентен следующим запросам:

```
select * from employees, dept_emp where employees.emp_no =  
dept_emp.emp_no;  
SELECT * FROM employees INNER JOIN dept_emp ON  
employees.emp_no = dept_emp.emp_no;  
SELECT * FROM employees INNER JOIN dept_emp USING (emp_no);
```

Внешние объединения

При внутреннем и естественном объединении соответствующие строки включаются в набор результатов. Внешнее объединение — это объединение, при котором в набор результатов включаются даже несовпадающие записи. В MySQL присутствуют два типа внешних объединений. Первый — это внешнее объединение `LEFT OUTER JOIN`, при котором в выходных данных присутствуют все записи из левой таблицы запроса на объединение, а также только соответствующие записи из правой таблицы запроса. Мы снова переключимся на другую базу данных с помощью следующего запроса:

```
USE Sakila;
```

Мы можем написать запрос в ANSI-стиле для `LEFT OUTER JOIN` следующим образом:

```
SELECT  
c.customer_id, c.first_name, c.last_name,  
a.actor_id, a.first_name, a.last_name  
FROM customer c  
LEFT OUTER JOIN actor a  
ON (c.last_name = a.last_name);
```

Также мы можем написать запрос следующим образом:

```

SELECT
c.customer_id, c.first_name, c.last_name,
a.actor_id, a.first_name, a.last_name
FROM customer c
LEFT JOIN actor a
ON (c.last_name = a.last_name);

```

В выходных данных мы можем следующим образом видеть значения NULL, соответствующие несовпадающей записи справа:

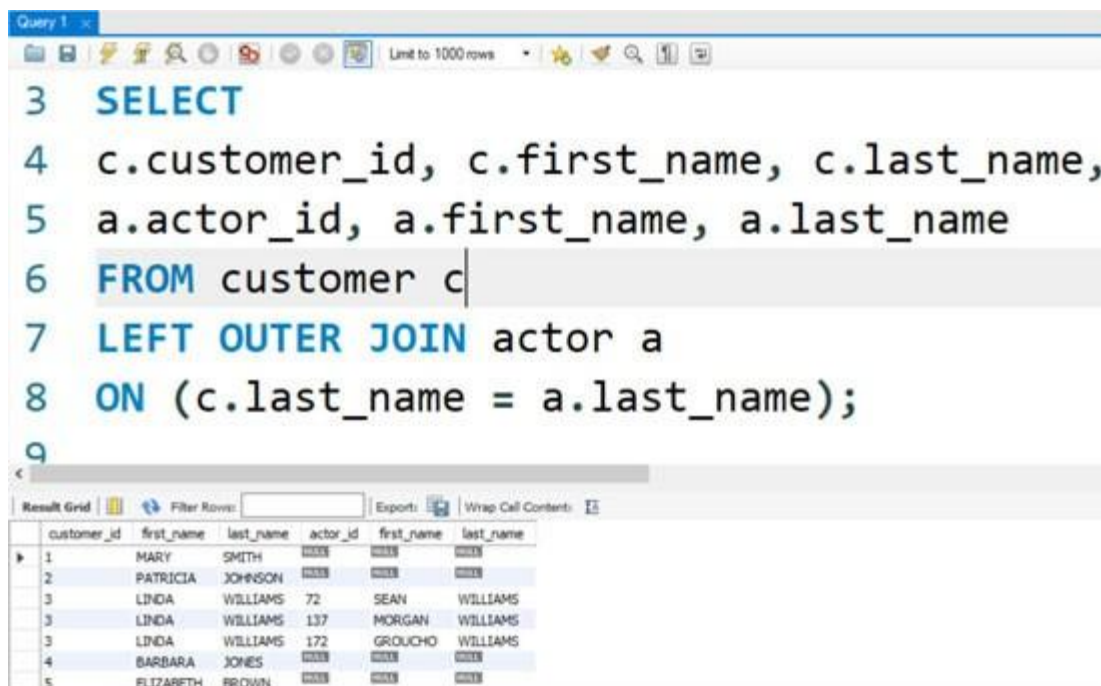


Рис. 7.2

Аналогично, в правом внешнем объединении RIGHT OUTER JOIN отображаются все записи из таблиц в правой части запроса и только соответствующие записи из левой таблицы. Мы можем написать запрос для RIGHT OUTER JOIN следующим образом:

```

SELECT
c.customer_id, c.first_name, c.last_name,
a.actor_id, a.first_name, a.last_name
FROM customer c
RIGHT OUTER JOIN actor a
ON (c.last_name = a.last_name);

```

Также мы можем написать запрос следующим образом:


```

SELECT
c.customer_id, c.first_name, c.last_name,
a.actor_id, a.first_name, a.last_name
FROM customer c
RIGHT JOIN actor a
ON (c.last_name = a.last_name);

```

В выходных данных мы можем следующим образом видеть значения NULL, соответствующие несовпадающей записи слева:

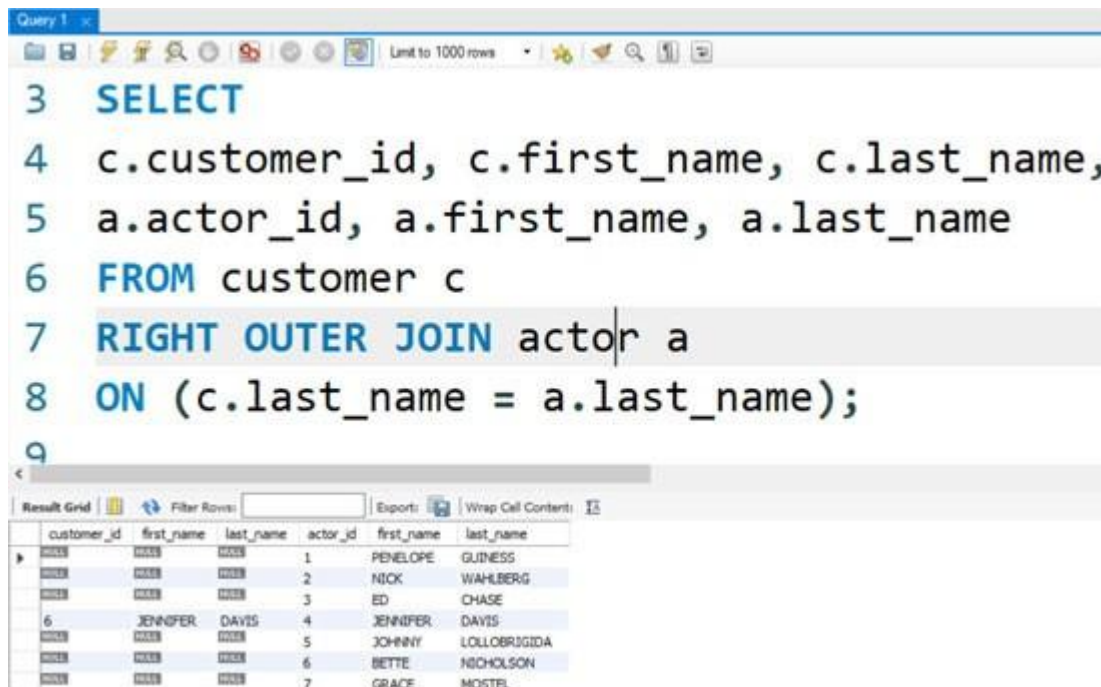


Рис. 7.3

Многие продукты баз данных, такие как база данных Oracle, поддерживают другой тип внешнего объединения, известный как полное внешнее объединение. MySQL и MariaDB не поддерживают этот тип объединения.

Объединение с несколькими условиями и несколькими источниками

До сих пор мы изучали и демонстрировали объединения с одним условием объединения. Мы также можем использовать следующим образом несколько условий в операторах объединения:

```

SELECT c.customer_id, c.first_name, c.last_name,
a.actor_id, a.first_name, a.last_name

```

```
FROM customer c
Inner JOIN actor a
ON (c.last_name = a.last_name AND c.first_name =
a.first_name);
```

В приведенном выше запросе мы видим, что присутствует два условия для внутреннего объединения. Аналогичным образом мы можем написать запрос с одним условием объединения и другим условием для оператора WHERE:

```
SELECT * FROM city INNER JOIN country
ON city.country_id = country.country_id
WHERE city.country_id IN ( 1, 4, 7);
```

Кроме того, мы видели запросы только с двумя таблицами в качестве источников. В запросе может фигурировать несколько источников. Правило таково: если у нас есть n источников, то объединение имеет $n-1$ условие. Ниже приведен пример такого запроса с тремя отдельными таблицами в качестве источников:

```
select first_name, last_name, title from film_actor, actor,
film
WHERE actor.actor_id = film_actor.actor_id AND
film_actor.film_id = film.film_id
```

Мы можем следующим образом написать тот же запрос, используя синтаксис ANSI:

```
select first_name, last_name, title from film_actor
JOIN actor USING (actor_id) JOIN film USING (film_id)
```

Самообъединение

Во многих случаях мы должны иметь одну таблицу или представление, выступающее в качестве нескольких источников. В таких ситуациях требуется особый тип объединения, известный как самообъединение. Ниже приведен пример:

```
SELECT a.customer_id, b.customer_id, a.first_name,
b.first_name, a.last_name, b.last_name
FROM customer a
INNER JOIN customer b
ON a.last_name = b.first_name;
```

В приведенном выше запросе мы видим, что используем одну таблицу с двумя разными псевдонимами и внутренним объединением. Мы даже можем использовать внешние объединения, если этого требует ситуация.

Неэквивалентное объединение

Мы можем использовать другие операторы в условии объединения. Такие объединения известны как неэквивалентные. Ниже приведен простой пример неэквивалентного объединения:

```
select * from city, country where city_id = 2
AND city.country_id < country.country_id
```

Мы должны использовать неравенство в определенных ситуациях. Это объединение идеально подходит для таких ситуаций.

Заключение

В этой главе мы изучили технику объединения данных из нескольких источников. Это известно как операция объединения. Мы изучили все типы объединений. Мы можем использовать объединения, когда нам нужно объединить данные из нескольких источников. Мы также можем использовать несколько источников и несколько условий в запросе на объединение.

В следующей главе мы подробно изучим подзапрос. Мы увидим несколько типов подзапросов и все сценарии использования подзапросов.

Что следует помнить

- Объединения объединяют данные из нескольких источников.
- Простые и естественные объединения объединяют данные на основе равенства.
- Неэквивалентные объединения объединяют данные с использованием операторов, отличных от равенства.
- MySQL и MariaDB не поддерживают полное внешнее объединение.
- В запросе на объединение с несколькими источниками, если имеется n источников, то существует $n-1$ условий для объединения.

Вопросы с выбором правильного ответа

1. Как по-другому называется простое объединение?
 - a) Внешнее объединение
 - b) Самообъединение
 - c) Внутреннее объединение
 - d) Неэквивалентное объединение
2. Сколько условий для объединения требуется, если запрос на объединение имеет n источников?
 - a) n
 - b) $n+1$
 - c) $n * n$
 - d) $n-1$

Ответы на Вопросы

1. c)
2. d)

Вопросы

1. Что такое объединения? Какие бывают виды объединений?
2. Объясните подробно на примерах внешнее объединение.
3. Как объединить несколько таблиц?
4. Объясните неэквивалентное объединение на примерах.

Ключевые термины

Объединения, эквивалентные объединения, неэквивалентные объединения, самообъединения, внешние объединения, объединения нескольких источников, внутреннее объединение, естественное объединение, простое объединение, перекрестное объединение, декартово произведение

ГЛАВА 8

Подзапросы

В прошлой главе мы изучили концепцию объединений. Мы увидели, что объединения помогают объединить данные из разных источников и создать единый набор результатов, который мы можем использовать для наших аналитических задач.

Мы также увидели различные категории объединений и объединения, предлагаемые MySQL и MariaDB. Мы узнали, что полное внешнее объединение недоступно в MySQL и MariaDB. Мы также изучили старый стиль тета и новый синтаксис стиля ANSI для написания объединений.

В этой главе мы изучим концепцию запроса внутри запроса. Это также известно как подзапросы. Мы это изучим и продемонстрируем весьма подробно. Мы увидим различные типы и применения концепции подзапроса. Подзапросы широко используются в различных сценариях. Они используются в основном в операторе `WHERE`. Иногда они также используются для создания таблиц (запросы `CTAS` – создание таблицы, которые мы увидим в следующей главе). Разработчику и специалисту по данным важно быть знакомым с подзапросами.

Структура

Ниже приведен подробный список тем, которые мы изучим и продемонстрируем с помощью SQL-запросов в этой главе:

- Подзапрос
- Связанный подзапрос

Цель

Мы подробно изучим и продемонстрируем все типы подзапросов. Прочитав эту главу, вы сможете успешно писать различные типы подзапросов для своих задач обработки данных и видеть разницу между простым и связанным подзапросом.

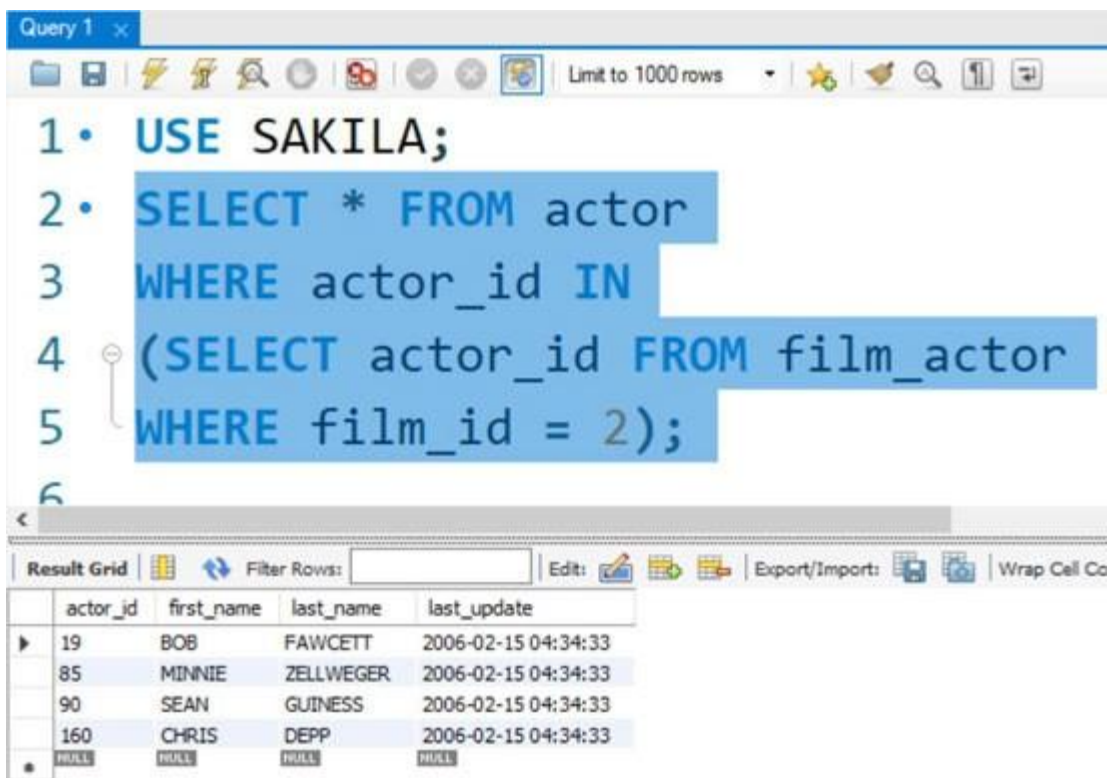
Подзапрос

Подзапрос означает запрос внутри запроса. Он также известен как вложенный запрос. Обычно он вложен в запросы `SELECT`, `DELETE`, `INSERT` или `UPDATE`. Подзапрос можно использовать в том месте, где разрешено размещение выражение. Давайте рассмотрим несколько примеров подзапроса, используемого в операторе выбора.

Для этого мы будем использовать базу данных Sakila:

```
USE SAKILA;
SELECT * FROM actor
WHERE actor_id IN
(SELECT actor_id FROM film_actor
WHERE film_id = 2);
```

В приведенном выше запросе мы видим два запроса. Основной запрос (также известный как внешний запрос) использует выходные данные, сгенерированные внутренним запросом. Таким образом, сначала выполняется внутренний запрос, а его выходные данные передаются оператору `WHERE` внешнего запроса для дальнейшей оценки набора результатов. Результат будет следующим:



The screenshot shows a SQL query editor window titled "Query 1". The query text is as follows:

```
1 • USE SAKILA;
2 • SELECT * FROM actor
3   WHERE actor_id IN
4   (SELECT actor_id FROM film_actor
5    WHERE film_id = 2);
6
```

The result set is displayed in a table below the query editor. The table has four columns: `actor_id`, `first_name`, `last_name`, and `last_update`. The data rows are:

actor_id	first_name	last_name	last_update
19	BOB	FAWCETT	2006-02-15 04:34:33
85	MINNIE	ZELLWEGER	2006-02-15 04:34:33
90	SEAN	GUINNESS	2006-02-15 04:34:33
160	CHRIS	DEPP	2006-02-15 04:34:33
NULL	NULL	NULL	NULL

Рис. 8.1

В результате показаны данные обо всех актерах, у которых идентификатор фильма `film_id` в базе данных равен 2.

У нас может быть много уровней подзапросов. Это известно как вложенные подзапросы:

```
SELECT * FROM actor
WHERE actor_id IN
(SELECT actor_id FROM film_actor
WHERE film_id =
(SELECT film_id FROM film
WHERE title = 'Ace Goldfinger')
);
```

В приведенном выше запросе мы видим, что есть два внутренних запроса. Самый внутренний запрос выполняется первым, а самый внешний запрос выполняется последним. Также мы можем использовать подзапрос следующим образом в качестве производной таблицы:

```
SELECT AVG(a) FROM
(SELECT
customer_id,
SUM(amount) a
FROM payment
GROUP BY customer_id) AS totals;
```

Здесь мы используем внутренний запрос для создания производной таблицы, которая выступает в качестве источника данных для внешнего запроса `SELECT`.

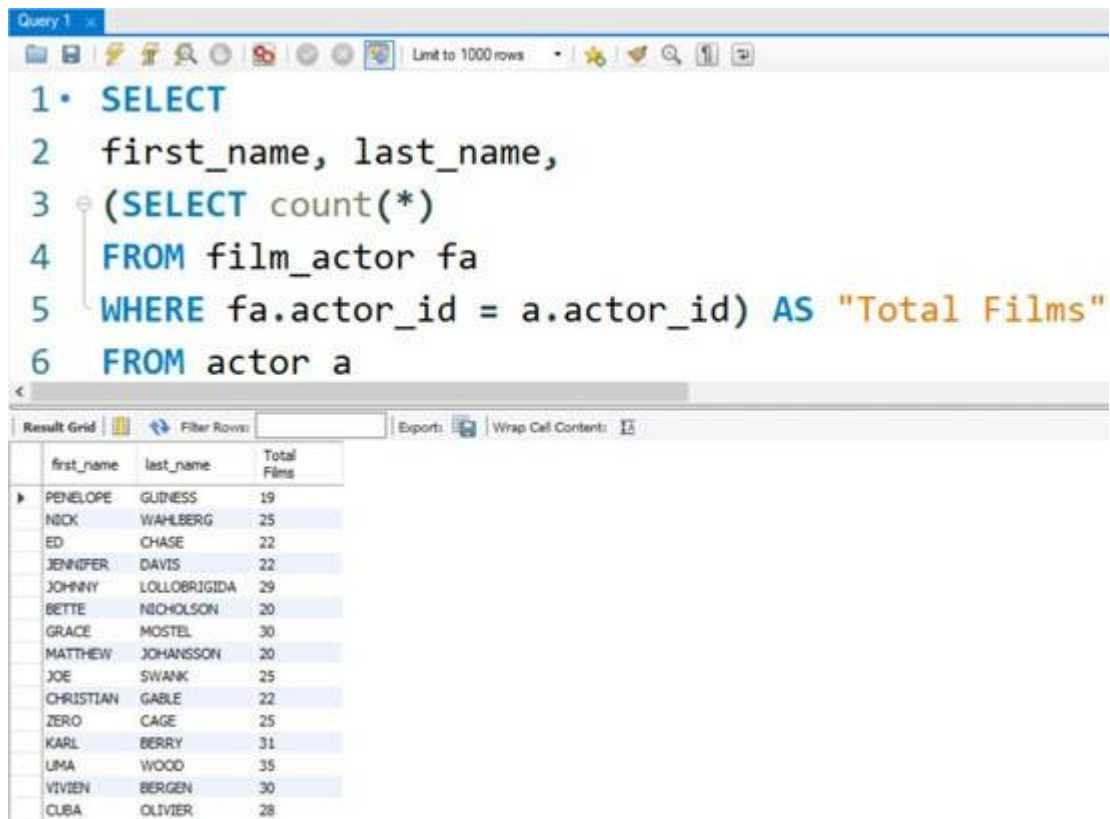
Связанный подзапрос

Связанный подзапрос также известен как синхронизированный подзапрос. Это подзапрос, который использует значения из внешнего запроса. В связанном подзапросе внешний и внутренний запросы выполняются на каждой итерации. Вот почему это может происходить очень медленно. Ниже приведен пример связанного подзапроса:

```
SELECT
first_name, last_name,
(SELECT count(*)
FROM film_actor fa
```

```
WHERE fa.actor_id = a.actor_id) AS "Total Films"  
FROM actor a
```

Результат будет следующим:



The screenshot shows a SQL query editor window titled "Query 1". The query text is as follows:

```
1 • SELECT  
2 first_name, last_name,  
3 (SELECT count(*)  
4 FROM film_actor fa  
5 WHERE fa.actor_id = a.actor_id) AS "Total Films"  
6 FROM actor a
```

Below the query editor is a "Result Grid" showing the output of the query. The grid has three columns: "first_name", "last_name", and "Total Films". The data is as follows:

first_name	last_name	Total Films
PENELOPE	GUINNESS	19
NOCK	WAHLBERG	25
ED	CHASE	22
JENNIFER	DAVIS	22
JOHNNY	LOLLOBRIGIDA	29
BETTE	NICHOLSON	20
GRACE	MOSTEL	30
MATTHEW	JOHANSSON	20
JOE	SWANK	25
CHRISTIAN	GABLE	22
ZERO	CAGE	25
KARL	BERRY	31
UMA	WOOD	35
VIVIEN	BERGEN	30
CUBA	OLIVIER	28

Рис. 8.2

Из-за низкой скорости связанных подзапросов разработчики обычно предпочитают использовать объединения для получения тех же данных за меньшее время. Подробнее об этом можно прочитать на странице <https://www.zentut.com/sql-tutorial/understanding-correlated-subquery/>.

Заключение

В этой главе мы изучили еще один метод объединения данных из нескольких источников. Он известен как подзапрос. Подзапросы специально используются, когда количество строк очень невелико. В противном случае на выполнение запроса потребуется много времени. Когда количество строк большое, мы используем объединения для объединения данных из нескольких источников.

В следующей главе мы изучим запросы языка определения данных **Data Definition Language (DDL)** и языка модификации данных **Data**

Modification Language (DML).

Что следует помнить

- ♦ Подзапросы объединяют данные из нескольких источников.
- ♦ Подзапросы медленнее, чем объединения.
- ♦ Подзапросы следует использовать, когда количество строк небольшое.
- ♦ Связанный подзапрос может быть очень дорогим с точки зрения времени обработки.

Вопросы с выбором верного ответа

1. Как по-другому называется подзапрос?
 - a) Внешнее объединение
 - b) Суперзапрос
 - c) Мегазапрос
 - d) Вложенный запрос
2. Как по-другому называется связанный подзапрос?
 - a) Синхронизированный запрос
 - b) Суперзапрос
 - c) Мегазапрос
 - d) Вложенный запрос

Ответы на вопросы

1. d)
2. a)

Вопросы

1. Что такое подзапрос?
2. Что такое связанный подзапрос?

Ключевые термины

Подзапрос, связанный подзапрос, вложенный запрос

ГЛАВА 9

DDL, DML и транзакции

В предыдущей главе мы изучили и продемонстрировали концепцию подзапросов. Мы узнали типы подзапросов и сценарии, в которых мы можем использовать подзапросы. Мы также узнали, что подзапросы требуют больших вычислительных затрат.

В этой главе мы узнаем, как создавать таблицы в MySQL и MariaDB. Мы также научимся выполнять запросы DDL и DML к таблицам. Мы изучим основы транзакций и способы их фиксации или отката.

Структура

Ниже приведен подробный список тем, которые мы изучим и продемонстрируем с помощью SQL-запросов в этой главе:

- Таблицы и словарь данных
- Операторы DDL
- Транзакции и операторы DML
- Операция усечения
- Создание таблицы как
- Ограничения
- Удаление базы данных

Цель

Прочитав эту главу, мы сможем создавать наши таблицы в схеме MySQL или MariaDB. Мы сможем выполнять запросы DDL и DML к таблицам. А также мы сможем подробно объяснить транзакцию.

Мы также увидим небольшое применение подзапроса и способы удаления базы данных.

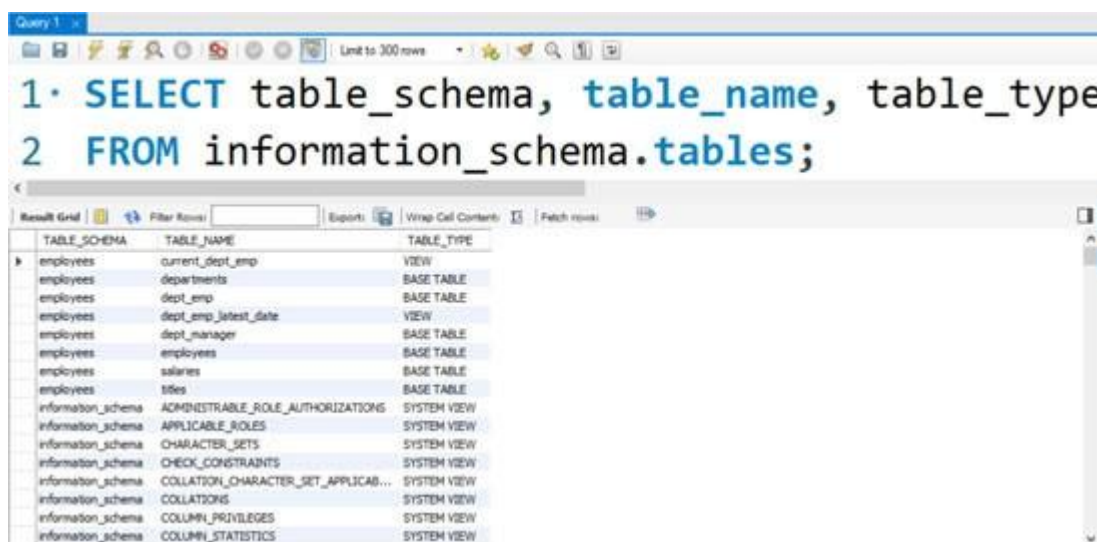
Таблицы и словарь данных

В реляционных базах данных таблица представляет собой набор данных, который организован в виде строк и столбцов. Таблица также называется двумерным массивом, а строка — кортежем. Таблицы обычно используются для хранения информации о схожих сущностях. Предположим, что есть учебное заведение, в котором учатся разные группы людей. В базе данных института мы можем иметь отдельные таблицы для сотрудников, студентов и отстающих.

Мы можем увидеть список таблиц, используя информационную схему следующим образом:

```
SELECT table_schema, table_name, table_type FROM
information_schema.tables;
```

Ниже показан результат:



```
1. SELECT table_schema, table_name, table_type
2. FROM information_schema.tables;
```

TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
employees	current_dept_emp	VIEW
employees	departments	BASE TABLE
employees	dept_emp	BASE TABLE
employees	dept_emp_latest_date	VIEW
employees	dept_manager	BASE TABLE
employees	employees	BASE TABLE
employees	salaries	BASE TABLE
employees	titles	BASE TABLE
information_schema	ADMINISTRABLE_ROLE_AUTHORIZATIONS	SYSTEM VIEW
information_schema	APPLICABLE_ROLES	SYSTEM VIEW
information_schema	CHARACTER_SETS	SYSTEM VIEW
information_schema	CHECK_CONSTRAINTS	SYSTEM VIEW
information_schema	COLLATION_CHARACTER_SET_APPLICAB...	SYSTEM VIEW
information_schema	COLLATIONS	SYSTEM VIEW
information_schema	COLUMN_PRIVILEGES	SYSTEM VIEW
information_schema	COLUMN_STATISTICS	SYSTEM VIEW

Рис. 9.1

Таблицы словарей данных в MySQL и MariaDB не видны. Таким образом, к ним нельзя получить прямой доступ с помощью SQL-запросов. MySQL и MariaDB поддерживают доступ к данным, хранящимся в таблицах словаря данных, посредством таблицы INFORMATION_SCHEMA и операторов SHOW. Мы увидим, как извлекать соответствующие данные из словаря данных по мере необходимости.

Операторы DDL

DDL - это язык определения данных. Операторы DDL создают, изменяют или удаляют объекты базы данных. В этой главе мы изучим довольно много операторов DDL. В данном разделе мы изучим операторы DDL, связанные с объектом таблица.

Мы будем использовать схему `employees`, чтобы продемонстрировать DDL, связанные с таблицами:

```
USE employees;
```

Следующий оператор создает таблицу в текущей выбранной схеме,

```
CREATE TABLE performance_records  
(  
  empno INT,  
  first_name VARCHAR(20),  
  last_name VARCHAR(20),  
  review_date DATETIME,  
  rating CHAR(5) DEFAULT NULL  
)
```

Мы знаем, что можем использовать следующий оператор для выбора записей из таблицы:

```
SELECT * FROM performance_records;
```

Мы можем увидеть структуру таблицы с помощью следующего оператора:

```
DESC performance_records;
```

Мы можем удалить всю структуру таблицы и все записи в таблице с помощью следующего оператора:

```
DROP TABLE IF EXISTS performance_records;
```

Теперь, поскольку структура таблицы не существует в схеме, следующие инструкции при выполнении будут возвращать ошибки:

```
DROP TABLE performance_records;  
SELECT * FROM performance_records;  
DESC performance_records;
```

Вы, должно быть, заметили, что мы присваиваем столбцам типы данных. Подробный список типов данных можно найти по следующим URL-адресам:

https://www.w3schools.com/sql/sql_datatypes.asp
<https://dev.mysql.com/doc/refman/8.0/en/data-types.html>

Давайте снова создадим исходную таблицу:

```
CREATE TABLE performance_records
(
empno INT,
first_name VARCHAR(20),
last_name VARCHAR(20),
review_date DATETIME,
rating CHAR(5) DEFAULT NULL
)
```

Мы можем переименовать объект таблицы следующим образом:

```
RENAME TABLE performance_records TO
performance_records_backup;
```

Теперь снова следующие запросы к исходной таблице вернут ошибки:

```
DROP TABLE performance_records;
SELECT * FROM performance_records;
DESC performance_records;
```

Мы можем увидеть структуру переименованной таблицы с помощью следующего оператора:

```
DESC performance_records_backup;
```

Теперь снова создайте для дальнейшей демонстрации исходную таблицу с помощью следующего запроса:

```
CREATE TABLE performance_records
(
empno INT,
first_name VARCHAR(20),
last_name VARCHAR(20),
review_date DATETIME,
```

```
rating CHAR(5) DEFAULT NULL
)
```

Мы можем добавить столбец с помощью следующего оператора и увидеть изменения в структуре таблицы:

```
ALTER TABLE performance_records ADD external_reviewer_name
VARCHAR(5) ;
DESC performance_records;
```

Мы можем изменить существующий столбец следующим образом:

```
ALTER TABLE performance_records MODIFY external_reviewer_name
VARCHAR(15) ;
DESC performance_records;
```

Мы можем удалить существующий столбец следующим образом:

```
ALTER TABLE performance_records DROP external_reviewer_name;
DESC performance_records;
```

Давайте удалим созданную нами резервную таблицу с помощью следующего оператора:

```
DROP TABLE IF EXISTS performance_records_backup;
```

Транзакции и операторы DML

Транзакция — это единица работы. Всякий раз, когда мы вносим какие-либо изменения в данные в базе данных, это называется транзакцией. Операторы, которые вызывают транзакции, известны как операторы **DML** или **Data Modification Language statements** (языка модификации данных). Прежде чем идти дальше, нам нужно отключить автоматическую фиксацию в нашем сеансе. В строке главного меню рабочей среды MySQL перейдите в меню «Query» и отключите опцию «Auto-Commit Transactions», как это показано на следующем изображении:

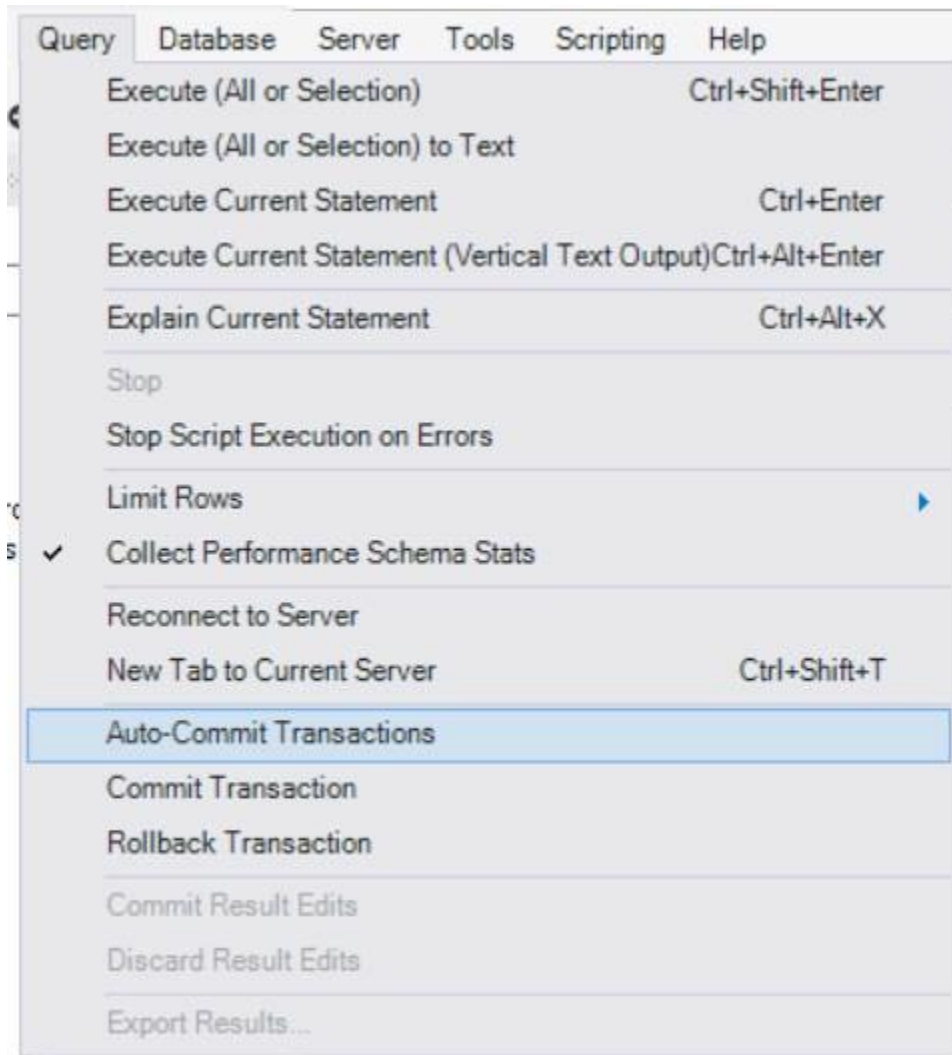


Рис. 9.2

Скоро мы увидим, что это значит. Самый первый оператор DML, который мы изучим, — это `INSERT`. Для этого упражнения мы будем использовать уже существующую таблицу. Запустите следующий запрос:

```
SELECT * FROM departments;
```

Запрос даст следующий результат:

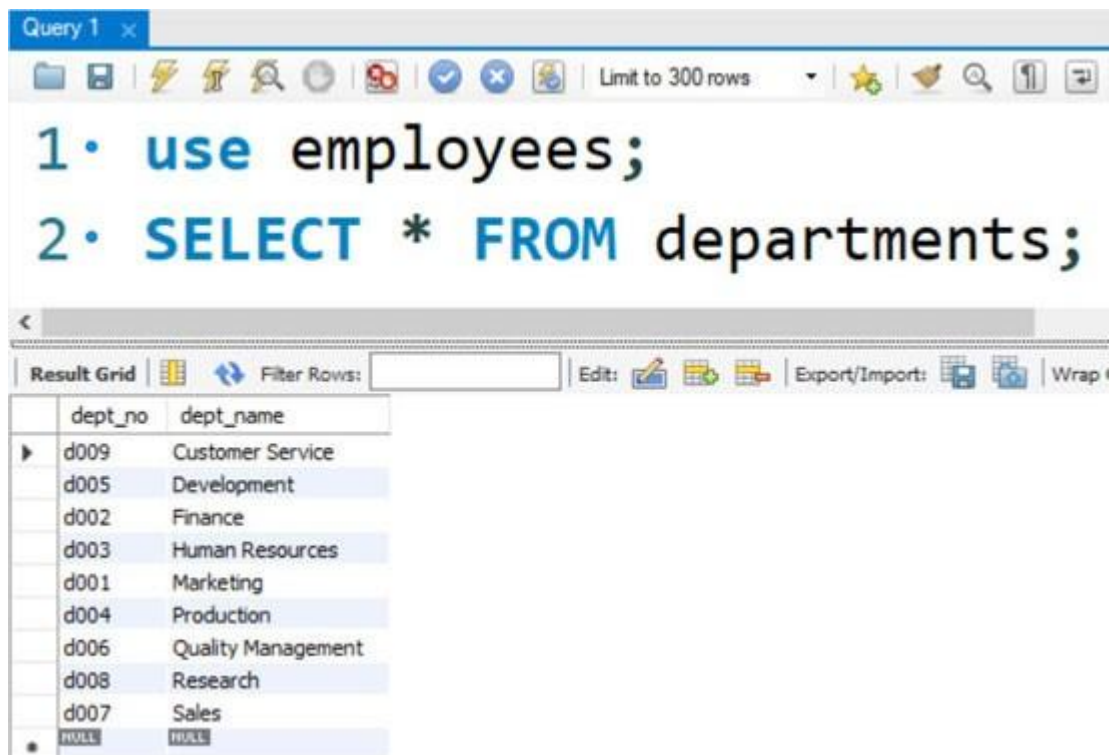


Рис. 9.3

Мы можем вставлять значения в столбцы, используя имя столбца в операторе следующим образом:

```
INSERT INTO departments (dept_no, dept_name) VALUES ('d099',  
'IT');
```

Мы можем вставлять значения, не упоминая наименования столбцов, но значения должны располагаться в том же порядке, что и столбцы в структуре таблицы. Ниже приведен пример:

```
INSERT INTO departments VALUES ('d100', 'Purchase');
```

После выполнения обоих запросов используйте следующий оператор, чтобы просмотреть значения в таблице:

```
SELECT * FROM departments;
```

О транзакции мы говорили в начале этого раздела. Всякий раз, когда мы впервые за сеанс запускаем оператор DML, он начинает транзакцию. Транзакция может быть совершена двумя способами. Мы либо фиксируем ее, либо откатываем. Всякий раз, когда мы иницируем транзакцию, если она не завершена, изменения не становятся постоянными. Итак, если мы попытаемся проверить значения в таблице из какого-либо другого сеанса,

они не будут показаны. Если мы хотим откатиться к состоянию начала транзакции, то используем следующую команду:

```
ROLLBACK  
;
```

Если мы сейчас запустим запрос выбора, он не отобразит добавленные строки. Давайте снова вставим новые строки и зафиксируем изменения, чтобы сделать их постоянными:

```
INSERT INTO departments (dept_no, dept_name) VALUES ('d099',  
'IT');  
INSERT INTO departments VALUES ('d100', 'Purchase');  
COMMIT;
```

Теперь, если мы увидим это из другого сеанса, мы сможем увидеть и добавленные строки. Таким образом, транзакция завершается после фиксации или отката.

Мы можем обновить существующие строки в таблице с помощью оператора UPDATE. Также мы можем обновить следующим образом всего одну строку:

```
UPDATE employees SET first_name = 'ASHWIN' WHERE emp_no =  
10018;
```

Оператор WHERE в приведенном выше запросе возвращает одну строку при использовании в запросе SELECT. Этот запрос, приведенный выше, обновляет одну строку. Мы также можем обновить несколько строк следующим образом:

```
UPDATE employees SET birth_date = birth_date + 100 WHERE  
emp_no IN (10001, 10002);
```

Мы можем завершить эту транзакцию либо с помощью операции фиксации, либо с помощью операции отката. Аналогичным образом мы можем удалить одну или несколько записей:

```
DELETE FROM employees WHERE emp_no = 10004;
```

Мы можем даже попытаться удалить все записи с помощью следующего запроса:

```
DELETE FROM  
employees;
```

Однако из-за безопасного режима обновления запрос не будет выполняться и вернет ошибку. Чтобы его все же выполнить, мы должны изменить наши глобальные параметры.

Операция усечения

Мы можем удалить все записи в таблице с помощью команды DDL, известной как TRUNCATE, следующим образом:

```
SELECT * FROM titles;
TRUNCATE TABLE titles;
SELECT * FROM titles;
```

Сначала выбираем все записи из таблицы. Затем мы ее обрезаем. После этого оператор SELECT не возвращает ни одной строки, поскольку все записи были обрезаны. Более того, оператор отката не отменит внесенные нами изменения. Мы можем попытаться сделать это следующим образом:

```
ROLLBACK
;
SELECT * FROM
titles;
```

Приведенный выше запрос ничего не вернет, поскольку операция отката не восстанавливает обрезанные строки. Это связано с тем, что усечение — это операция DDL, а все операции DDL (в отличие от операций DML) по своей природе являются автоматическими.

Создание таблицы

Теперь мы изучим и продемонстрируем специальный оператор DDL **CREATE TABLE AS**. Его также называют **CTAS**. Он использует простой подзапрос для создания структуры таблицы и заполнения ее существующими данными. Ниже приведен пример:

```
CREATE TABLE EMPLOYEES_BACKUP AS (SELECT * FROM EMPLOYEES)
```

Ограничения

Давайте изучим концепцию ограничений. В контексте СУБД ограничения

— это правила ограничений, применяемые к столбцам таблицы. Эти правила определяют данные, которые могут храниться в таблицах.

Существует два типа ограничений. Это ограничения уровня таблицы и ограничения уровня столбца. Ограничения уровня столбца применяются только к тому столбцу, для которого они определены. Ограничения уровня таблицы применяются ко всей таблице. Наименования ограничений могут либо задаваться системой, либо задаваться нами самими.

Ниже приведены различные ограничения, которые мы можем накладывать на уровне таблицы или столбца:

1. **Primary Key:** Primary Key или первичный ключ — это столбец или набор столбцов, который используется для уникальной идентификации записи в таблице. Ключевые столбцы должны иметь уникальные, а не нулевые значения.
2. **NOT NULL:** Это означает, что столбец не может содержать значение NULL.
3. **UNIQUE:** Означает, что все значения в столбце должны быть уникальными. Оно может иметь значение NULL.
4. **CHECK:** Значения в столбце должны быть допустимыми для логического выражения.
5. **DEFAULT:** Означает, что когда мы вносим запись в таблицу, столбец будет иметь значение, принимаемое по умолчанию, если мы не укажем его явно.
6. **Foreign Key:** Означает, что столбец может иметь значение из набора значений, хранящихся в каком-либо другом столбце в той же или другой таблице.

Мы можем сослаться на следующие URL-адреса, чтобы прочитать и применить ограничение в MySQL:

<https://www.w3resource.com/mysql/creating-table-advance/constraint.php>

Удаление базы данных

Мы можем удалить базу данных (схему) с помощью следующей команды:

```
DROP DATABASE IF EXISTS employees;
```

Как и при любой команде удаления, как только мы удаляем базу данных, все объекты в этой базе данных удаляются безвозвратно. Для работы с удаленной базой данных необходимо выполнить действия по ее установке из внешних источников, как мы обсуждали ранее.

Заключение

В этой главе мы изучили и продемонстрировали концепции DDL и DML. Теперь нам удобно создавать новую таблицу, изменять ее и удалять из схемы.

Также мы можем выполнять к ним различные запросы DDL и DML и подробно объяснить транзакции. Теперь мы знаем, как использовать подзапрос для создания таблицы из существующей таблицы и для удаления базы данных.

В следующей главе мы рассмотрим концепцию представлений. Мы изучим различные типы представлений в SQL и покажем, как их создавать с помощью MySQL и MariaDB.

Что следует помнить

- Нам следует отключить автоматическую фиксацию перед работой с операторами DML.
- Операторы DDL по умолчанию являются автоматически фиксируемыми.
- TRUNCATE — это команда DDL для удаления всех строк из таблицы. Ее нельзя откатить назад.
- Мы можем использовать подзапросы для создания таблицы из существующих табличных структур.

Вопросы с выбором правильного ответа

1. TRUNCATE — это команда DDL.
 - a) Истина
 - b) Ложь
2. Какой оператор мы используем, чтобы сделать изменения, внесенные операторами DML, доступными для базы данных?

a) ROLLBACK

b) FINAL

c) DONE

d) COMMIT

Ответы на вопросы

1. *a)*

2. *d)*

Вопросы

1. Что такое оператор DDL?

2. Что такое транзакция? Объясните транзакции с помощью операторов DML, COMMIT и ROLLBACK.

Ключевые термины

COMMIT, транзакции, ROLLBACK, DDL, DML, CTAS, TRUNCATE, ограничения

ГЛАВА 10

Представления

В предыдущей главе мы изучили концепцию таблиц, словарей данных, операторов DDL, операторов DML и операторов управления транзакциями. Мы увидели, как создавать таблицы, как вставлять, обновлять и удалять данные и как управлять транзакциями. Также мы изучили основы словаря данных в базе данных.

В этой главе мы изучим и продемонстрируем еще один важный тип объектов базы данных. Это известно как концепция представления.

Структура

Ниже приведен подробный список тем, которые мы изучим и продемонстрируем с помощью SQL-запросов в этой главе:

- Концепция представлений
- Простые представления
- Сложные представления

Цель

Мы изучим и продемонстрируем все типы представлений в MySQL и MariaDB. Прочитав эту главу, вы сможете создавать представления в соответствии с требованиями. Мы также сможем определить сценарии, в которых мы можем использовать представления.

Концепция представлений

Представление — это оператор SQL со связанным именем. Это именованный запрос, хранящийся в каталоге базы данных, который позволяет нам обращаться к нему позже, когда нам это необходимо. Если говорить совсем конкретно, представление — это именованная виртуальная таблица. В этой главе мы будем использовать базу данных sakila:

```
USE SAKILA;
```

В предыдущей главе мы узнали о словарях и каталогах данных. Давайте запросим словарь данных для получения списка всех представлений с помощью следующего оператора:

```
SHOW FULL TABLES IN sakila WHERE TABLE_TYPE LIKE 'VIEW';
```

Ниже приводится результат:

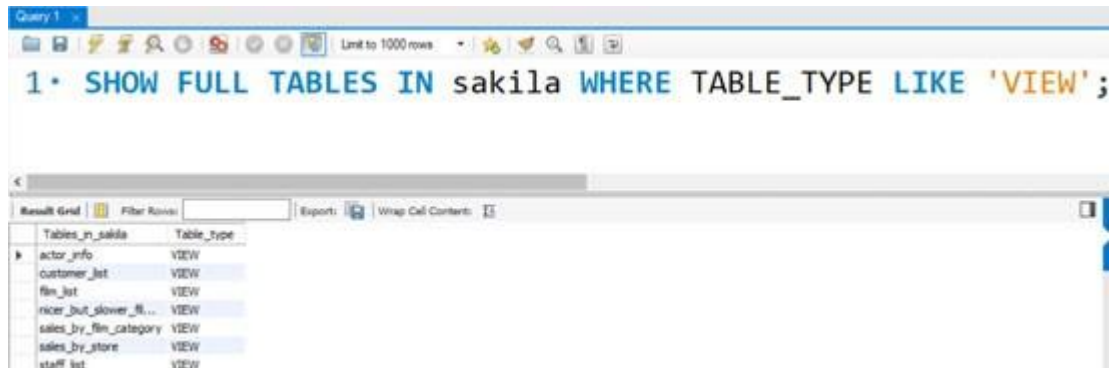


Рис. 10.1

Эти выходные данные отображают список всех представлений текущей базы данных (схеме). Мы можем увидеть данные представления с помощью следующего запроса:

```
SELECT * FROM actor_info;
```

Ниже приводится результат:

The screenshot shows a database query tool interface. At the top, a query editor displays the SQL statement: `SELECT * FROM actor_info;`. Below the editor, a toolbar includes icons for various actions and a 'Limit to 1000 rows' dropdown. The main area shows a 'Result Grid' with a table of data. The table has four columns: `actor_id`, `first_name`, `last_name`, and `film_info`. The data is presented as a list of 23 rows, each representing an actor and their associated film information.

	actor_id	first_name	last_name	film_info
▶	1	PENELOPE	GUINNESS	Animation: ANACONDA CONFESSIONS; Childre...
	2	NICK	WAHLBERG	Action: BULL SHAWSHANK; Animation: FIGHT J...
	3	ED	CHASE	Action: CADDYSHACK JEDI, FORREST SONS; Cl...
	4	JENNIFER	DAVIS	Action: BAREFOOT MANCHURIAN; Animation: A...
	5	JOHNNY	LOLLOBRIGIDA	Action: AMADEUS HOLY, GRAIL FRANKENSTEIN...
	6	BETTE	NICHOLSON	Action: ANTITRUST TOMATOES; Animation: BIK...
	7	GRACE	MOSTEL	Action: BERETS AGENT, EXCITEMENT EVE; Anim...
	8	MATTHEW	JOHANSSON	Action: CAMPUS REMEMBER, DANCES NONE; A...
	9	JOE	SWANK	Action: PRIMARY GLASS, WATERFRONT DELIVE...
	10	CHRISTIAN	GABLE	Action: LORD ARIZONA, WATERFRONT DELIVE...
	11	ZERO	CAGE	Action: DANCES NONE, HANDICAP BOONDOCK...
	12	KARL	BERRY	Action: STAGECOACH ARMAGEDDON; Animatio...
	13	UMA	WOOD	Action: ANTITRUST TOMATOES, CLUELESS BUC...
	14	VIVIEN	BERGEN	Action: DRIFTER COMMANDMENTS, EXCITEME...
	15	CUBA	OLIVIER	Action: MONTEZUMA COMMAND, WEREWOLF L...
	16	FRED	COSTNER	Action: EASY GLADIATOR, ENTRAPMENT SATIS...
	17	HELEN	VOIGHT	Action: SIDE ARK; Animation: CLASH FREDDY, ...
	18	DAN	TORN	Action: REAR TRADING; Animation: EARLY HOM...
	19	BOB	FAWCETT	Action: DARN FORRESTER; Animation: DARES P...
	20	LUCILLE	TRACY	Action: REAR TRADING; Animation: DOORS PR...
	21	KIRSTEN	PALTROW	Action: DRIFTER COMMANDMENTS, LORD ARIZ...
	22	ELVIS	MARX	Action: BAREFOOT MANCHURIAN, CADDYSHAC...
	23	SANDRA	KILMER	Action: BULL SHAWSHANK, DARN FORRESTER, ...

*Рис.
10.2*

Мы можем увидеть базовый запрос для представления с помощью следующих двух операторов:

```
SHOW CREATE VIEW actor_info;
```

```
SELECT VIEW_DEFINITION
FROM INFORMATION_SCHEMA.VIEWS
```

```
WHERE TABLE_SCHEMA = 'sakila'  
AND TABLE_NAME = 'actor_info';
```

Мы научимся создавать подобные представления в следующих двух разделах.

Простые представления

Простые представления не применяют никаких преобразований к столбцам базового запроса. Следующий запрос создает простое представление:

```
CREATE OR REPLACE VIEW address_view AS SELECT  
address, postal_code FROM address;
```

Приведенный выше запрос показывает два столбца таблицы. Мы можем увидеть данные представления с помощью следующего запроса:

```
SELECT * FROM  
address_view;
```

Мы можем даже переопределить представление следующим образом,

```
CREATE OR REPLACE VIEW address_view AS SELECT  
address, postal_code FROM address WHERE postal_code  
IS NOT NULL;
```

В приведенном выше запросе мы добавили в представление оператор `WHERE`. Простые представления в основном создаются из одной таблицы и не содержат сложного запроса. Не обязательно производить объединение. Простые представления не содержат групповых функций, групп данных, `GROUP BY` и `DISTINCT`. Мы можем выполнять в простом представлении операции DML над базовыми таблицами. Мы можем выбирать данные из представления с помощью простого запроса `SELECT`.

Сложные представления

Сложные представления — это представления, которые могут иметь сложные базовые запросы с объединениями, групповыми функциями,

группами данных, `GROUP BY` и `DISTINCT`. Мы не можем выполнять операции DML над базовыми таблицами. Ниже приведен пример сложного представления:

```
CREATE OR REPLACE VIEW city_country_info AS SELECT city,
country FROM city INNER JOIN country USING (country_id);
```

Мы можем получить данные из сложного представления с помощью простого запроса `SELECT` следующим образом:

```
SELECT * FROM city_country_info;
```

Мы можем увидеть основной запрос сложного представления с помощью следующего оператора:

```
SHOW CREATE VIEW city_country_info;
```

Заключен ие

В этой главе мы изучили теорию и демонстрацию представлений. Мы видели различные типы представлений и словари данных, связанные с представлениями. Мы узнали, что представления — это виртуальные таблицы, созданные вместо очень длинных и часто используемых запросов, чтобы сэкономить время на написании запросов с нуля. Мы также можем использовать представления в качестве источников данных для других запросов. Теперь нам комфортно работать с представлениями, и мы можем при необходимости их создавать.

В следующей главе мы узнаем, как использовать базы данных MySQL и MariaDB с Python и Pandas.

Что следует помнить

- Очень большой и сложный оператор `SELECT` можно преобразовать в представление, поэтому нам больше не придется вводить этот оператор. Мы можем просто запустить запрос `SELECT` для представления.
- Простые представления позволяют выполнять операции DML с базовой таблицей.
- Сложные представления не позволяют выполнять операции

DML с базовой таблицей.

Вопросы с выбором правильного ответа

1. Позволяет ли простое представление выполнять операции DML с базовой таблицей?
 - a) Да
 - b) Нет
2. Позволяет ли сложное представление выполнять операции DML с базовой таблицей?
 - a) Да
 - b) Нет
3. Простое представление не допускает одно из следующего в базовом запросе.
 - a) WHERE
 - b) ORDER BY
 - c) JOIN
 - d) Column Alias

Ответы на вопросы

1. a)
2. b)
3. c)

Вопросы

1. Что такое представление?
2. Подготовьте таблицу различий между простыми и сложными представлениями.

Ключевые термины

Представление, простое представление, сложное представление

ГЛАВА 11

Python 3, MySQL и Pandas

В предыдущей главе мы изучили и продемонстрировали концепцию представлений. Мы видели, что представления — это виртуальные таблицы, и мы создаем их, когда у нас есть повторяющийся запрос. Мы также узнали, что существуют две категории представлений: простые и сложные. Мы также рассмотрели несколько сценариев, в которых мы можем использовать представления.

Язык программирования Python — один из наиболее часто используемых языков программирования нашей эпохи. Python — один из наиболее предпочтительных языков для науки о данных.

В этой главе мы узнаем, как связать язык программирования Python 3 с MySQL и MariaDB. Мы также узнаем, как связать его с популярной библиотекой обработки данных на Python 3, известной как Pandas. Это даст всем читателям четкое представление о роли реляционных баз данных в области науки о данных.

Структура

Ниже приведен подробный список тем, которые мы изучим и продемонстрируем с помощью SQL-запросов в этой главе:

- Установка Python 3
- Запуск Python 3
- Программирование MySQL и Python 3
- MySQL и Pandas

Цель

После прочтения этой главы мы сможем установить Python 3 в среде ОС Windows. Мы также сможем запускать программы Python 3 из командной строки ОС и с помощью редактора IDLE. Мы изучим основные функции IDLE и интерактивный режим Python 3. Мы также сможем связать MySQL с Python 3 и загрузить записи таблицы в фрейм

данных Pandas.

[Установка Python 3](#)

Python — самый популярный язык программирования на момент написания этой книги (2020-е годы). И в обозримом будущем он будет также одним из наиболее широко используемых языков программирования. Мы можем проверить популярность языков программирования в сети Интернет по следующим URL-адресам:

- ♦ <http://pypl.github.io/PYPL.html>
- ♦ <https://www.tiobe.com/tiobe-index/>

Язык программирования Python имеет два основных релиза: Python 2 и Python 3. С начала 2020 года выпуск Python 2 был прекращен, и только Python 3 остается в активной разработке.

Все дистрибутивы Linux поставляются с предустановленным Python 3. Нам просто нужно установить редактор для программирования на Python. Интегрированная среда разработки и обучения **Integrated Development and Learning Environment (IDLE)** — один из самых популярных редакторов для Python 3, который предустановлен во многих дистрибутивах Linux. Мы можем установить его в Debian и его производные, выполнив следующую команду в командной строке:

```
sudo apt-get install idle3
```

При этом будет установлен IDLE3, версия IDLE3 для Python 3. Мы часто можем найти его в меню дистрибутива Linux или среди ярлыков. Мы также можем запустить его из командной строки Linux с помощью следующей команды:

```
idle
```

Позже мы увидим, как работать с IDLE. Теперь давайте установим Python 3 в среде Windows. Python 3 не предустановлен в Windows. Нам нужно посетить домашнюю страницу Python www.python.org и загрузить установочный файл из раздела загрузок. Веб-сайт определяет операционную систему и предлагает правильный выбор для загрузки установочного файла Python 3 последней версии. На момент написания этой книги это версия 3.8.3. В будущем будет что-то еще, но процесс установки не будет сильно отличаться. Ниже

приведен скриншот страницы загрузки:

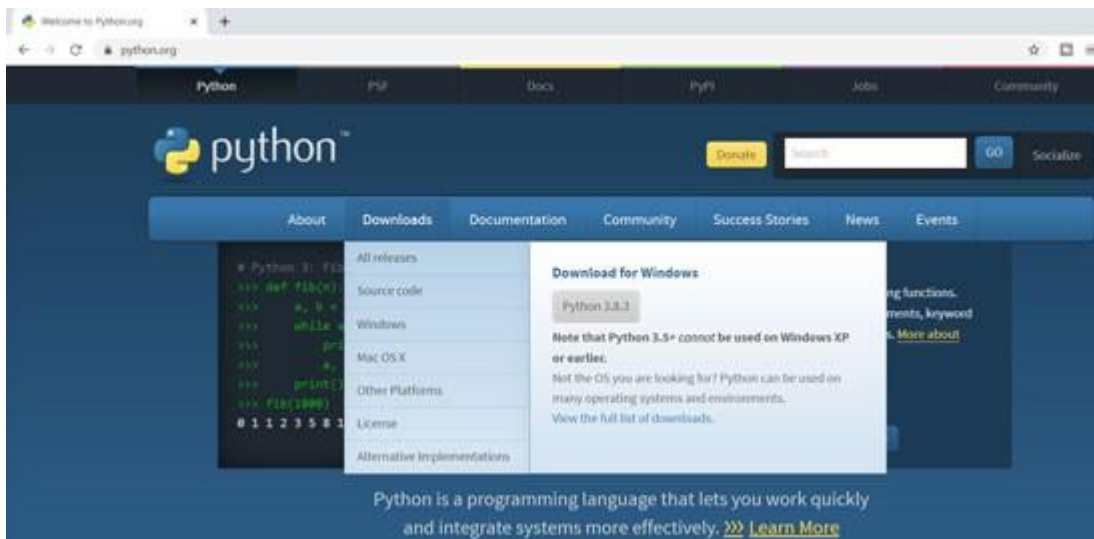


Рис. 11.1

После завершения загрузки мы сможем найти загруженный установочный файл в каталоге «Загрузки» нашего пользователя в среде Windows. Дважды щелкните его, чтобы запустить, появится следующее окно:



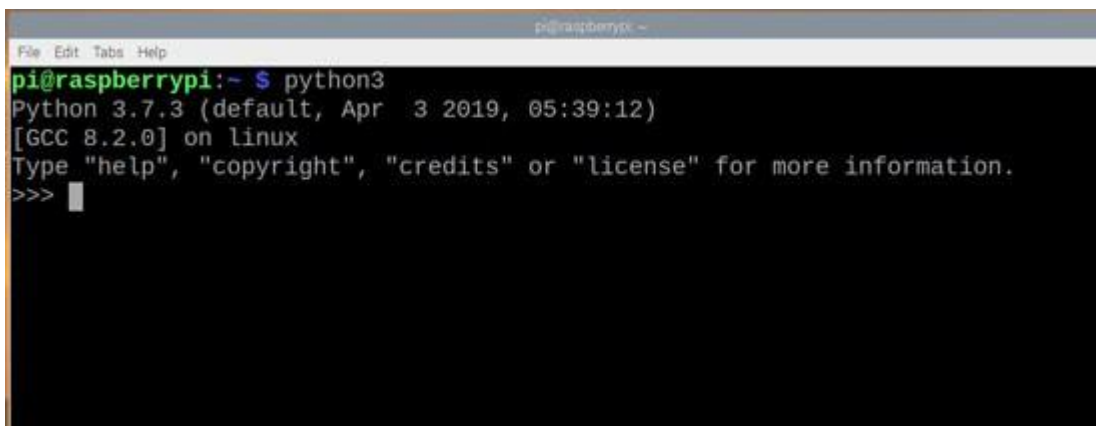
Рис. 11.2

В нижней части окна есть два флажка. Пометьте оба из них, щелкнув на их. Первый флажок устанавливает программу запуска, а второй добавляет

Python 3 в переменную PATH среды Windows. Затем выберите опцию **Install Now**. Он запросит у нас учетные данные администратора, предоставьте их для продолжения установки. Установка продолжится, а после завершения отобразится окно успешного завершения.

Запуск программы Python 3

Python 3 имеет два режима выполнения операторов. Первый — интерактивный режим. Это аналог командной строки операционной системы. Мы можем вводить операторы Python 3 в командной строке Python 3 и запускать их один за другим. Есть два способа вызвать интерактивный режим. Мы можем запустить командную строку ОС (cmd в Windows и lxterminal в Linux) и запустить команду `python3` в Linux и `python` в Windows, чтобы запустить Python 3 в интерактивном режиме. Ниже приведен скриншот Python 3, работающего в интерактивном режиме в командной строке Raspberry Pi:

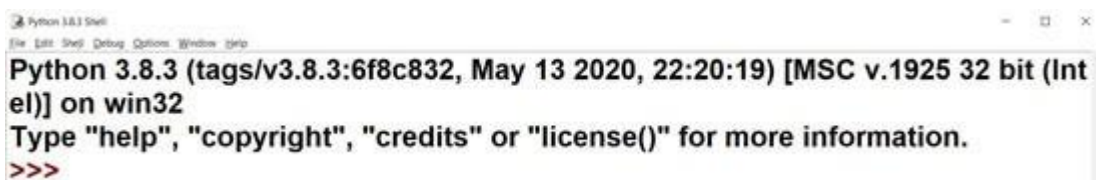


```
pi@raspberrypi:~$ python3
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Рис. 11.3

Другой способ — запустить IDLE. Ранее мы уже видели, как запустить его в среде Linux. Мы можем запустить его и в среде Windows, выполнив поиск в поле поиска Windows.

Интерфейс интерактивного режима Python в командной строке и IDLE одинаков в разных операционных системах. Ниже приведен скриншот IDLE в интерактивном режиме:



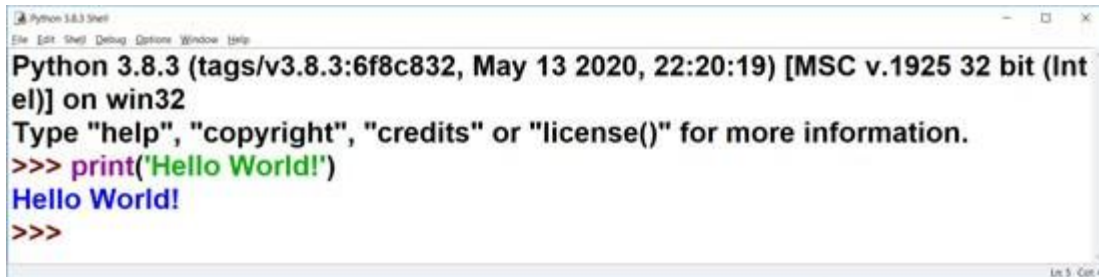
```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```


Рис. 11.4

Мы можем вводить сюда операторы один за другим и нажимать *Enter*, чтобы выполнить их. Введите следующий оператор в интерактивной строке и нажмите клавишу *Enter* на клавиатуре, чтобы выполнить его:

```
print('Hello  
World!')
```

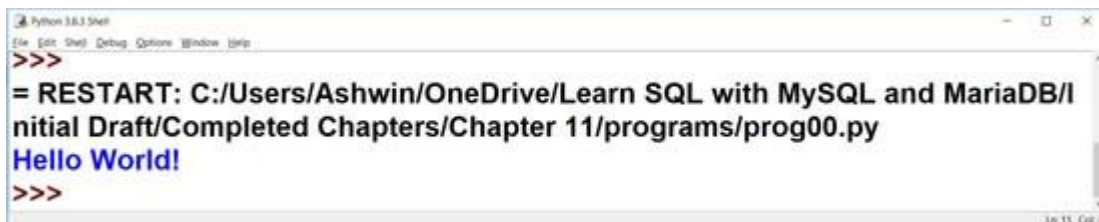
Результат будет следующим:



```
Python 3.8.3 Shell
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hello World!')
Hello World!
>>>
```

Рис. 11.5

Другой режим известен как режим сценария. В этом режиме мы создаем файл и сохраняем его с расширением `.py` (на самом деле расширение не имеет значения, если в нем есть код Python 3). Затем мы наполняем его операторами кода. Мы можем сделать это с помощью IDLE или любого другого редактора по нашему выбору. В IDLE выберите в меню **File > New File**. Он создает новый пустой файл. Сохраните его как `prog00.py`. Мы можем добавить в этот файл оператор, который мы выполнили ранее, сохранить его и запустить, используя меню **Run > Run Module**. Мы также можем запустить его, нажав клавишу *F5* на клавиатуре. Он будет выполняться в интерактивном запросе IDLE, а ниже приводится результат его выполнения:



```
Python 3.8.3 Shell
>>>
= RESTART: C:/Users/Ashwin/OneDrive/Learn SQL with MySQL and MariaDB/Initial Draft/Completed Chapters/Chapter 11/programs/prog00.py
Hello World!
>>>
```

Рис. 11.6

Мы также можем запустить программу из командной строки, используя исполняемый файл Python 3. Из командной строки перейдите в каталог, в котором хранится программа, и выполните следующую команду в `cmd` в

среде Windows:

```
python  
prog00.py
```

В среде Linux нам нужно запустить следующую команду в Ixterminal:

```
python3 prog00.py
```

Мы также можем запустить программу в командной строке из любого места, если укажем абсолютный путь к исполняемому файлу Python 3 следующим образом:

```
python "C:\Users\Ashwin\OneDrive\Learn SQL with MySQL and  
MariaDB\Initial Draft\Completed Главас\Глава  
11\programs\prog00.py"
```

В среде Linux мы можем запустить следующую команду:

```
python3 /home/pi/book/chapter11/prog00.py
```

Она запустит программу в командной строке ОС и распечатает выходные данные.

MySQL и Python 3

Мы можем запускать операторы MySQL или MariaDB из программ Python 3. Для этого нам нужно установить библиотеку, известную как `pymysql`. Мы можем установить его с помощью `pip`. `pip` — это менеджер пакетов Python, а это означает, что `pip` устанавливает пакеты (или `pip` устанавливает Python). Это рекурсивная аббревиатура. Мы можем запустить следующую команду в командной строке операционной системы, чтобы установить `pymysql` и другую необходимую пакетную криптографию:

```
pip3          install  
pymysql  
pip3 install cryptography
```

После завершения установки создайте нового пользователя и предоставьте ему все привилегии для всех схем, используя следующие инструкции SQL:

```
CREATE USER 'testuser'@'localhost' IDENTIFIED BY 'test123';  
GRANT ALL PRIVILEGES ON *.* TO 'testuser'@'localhost';
```

Теперь мы можем написать следующую программу на Python 3 для

проверки соединения:

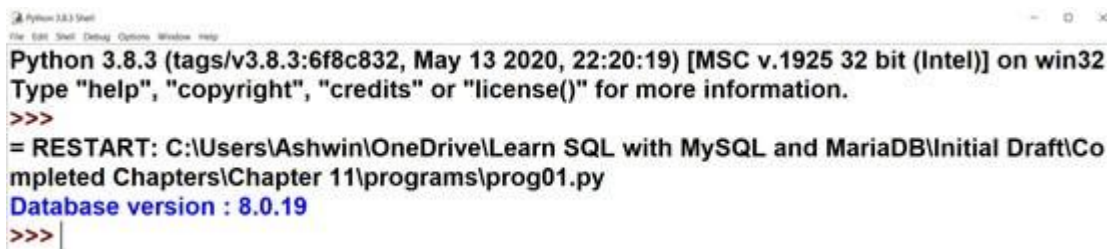
```
import pymysql
db =
pymysql.connect("localhost","testuser","test123","sakila",330
6)

cursor = db.cursor()
sql = "SELECT VERSION()"
cursor.execute(sql)
data = cursor.fetchone()
print("Database version : %s" % data)
db.close()
```

В первой строке мы импортируем необходимую библиотеку `pymysql`. Функция `pymysql.connect()` принимает `hostname` (имя хоста), `username` (имя пользователя), `password` (пароль), `database name` (наименование базы данных), а также номер порта в качестве аргументов и возвращает объект соединения. С помощью возвращенного объекта мы создаем объект курсора, вызывая функцию `db.cursor()`. Мы используем тот же курсор для выполнения строки, содержащей запрос SQL, с помощью оператора `cursor.execute(sql)`. Затем мы можем получить результат с помощью оператора `cursor.fetchone()`. После получения записи из набора результатов и ее отображения мы закрываем соединение с помощью кода `db.close()`.

Программа выше выбирает версию базы данных и отображает ее в интерактивной подсказке Python 3.

Ниже приведен скриншот с результатами:



```
Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Ashwin\OneDrive\Learn SQL with MySQL and MariaDB\Initial Draft\Completed Chapters\Chapter 11\programs\prog01.py
Database version : 8.0.19
>>>
```

Рис. 11.7

Теперь давайте рассмотрим следующий пример оператора DDL на Python 3:

```
import pymysql
db =
```

```

pymysql.connect("localhost","testuser","test123","sakila",330
6)
cursor = db.cursor()
try:
    sql = """CREATE TABLE TEST_TABLE (
    FIRST_NAME CHAR(20) NOT NULL,
    LAST_NAME CHAR(20) NOT NULL,
    AGE INT,
    SEX CHAR(1),
    INCOME FLOAT)"""
    cursor.execute(sql)
    print("The table created successfully!")
except:
    print("The table already exists in the database!")
db.close()

```

В приведенной выше программе мы запускаем оператор DDL. Если оператор выполняется успешно, отображается сообщение об успехе. Если мы попытаемся запустить программу дважды, она уже нашла объект в базе данных и вернет сообщение об ошибке.

Мы можем добавить записи в эту таблицу с помощью следующего кода:

```

import pymysql
db =
pymysql.connect("localhost","testuser","test123","sakila",330
6)
cursor =
db.cursor()
sql1 = """INSERT INTO TEST_TABLE (FIRST_NAME,
    LAST_NAME, AGE, SEX, INCOME)
    VALUES('Ashwin', 'Pajankar', 32, 'M', 1000)"""
sql2 = """INSERT INTO TEST_TABLE (FIRST_NAME,
    LAST_NAME, AGE, SEX, INCOME)
    VALUES('Thor', 'Odinson', 35, 'M', 2000)"""
sql3 = """INSERT INTO TEST_TABLE (FIRST_NAME,
    LAST_NAME, AGE, SEX, INCOME)
    VALUES('Tony', 'Stark', 40, 'M', 40000)"""
sql4 = """INSERT INTO TEST_TABLE (FIRST_NAME,
    LAST_NAME, AGE, SEX, INCOME)
    VALUES('Jane', 'Foster', 32, 'F', 3000)"""

```

```

try
:
    cursor.execute(sql1
    )
    cursor.execute(sql2
    )
    cursor.execute(sql3
    )
    cursor.execute(sql4
    )

    db.commit()
    print("The records inserted successfully!")
except:
    db.rollback()
    print("The records were not inserted!")
db.close()

```

В приведенной выше программе мы можем зафиксировать транзакцию, если все добавления были произведены успешно, в противном случае мы можем произвести откат.

Мы можем получить записи из таблицы следующим образом:

```

import pymysql
db =
pymysql.connect("localhost","testuser","test123","sakila",330
6)
cursor = db.cursor()
sql = "SELECT * FROM TEST_TABLE"
try:
    cursor.execute(sql)
    resultset = cursor.fetchall()
    for row in resultset:
        fname = row[0]
        lname = row[1]
        age = row[2]
        sex = row[3]
        income = row[4]
        print("%s %s, %d, %s, %d" %(fname, lname, age, sex,
income))

```

```
print("The records fetched successfully!")

except:
    print("The records were not fetched!")
db.close()
```

В приведенной выше программе мы используем `cursor.fetchall()` для извлечения всех записей в переменную, а затем циклически обрабатываем эту переменную для печати всех данных.

Мы можем обновить записи следующим образом:

```
import pymysql
db =
pymysql.connect("localhost","testuser","test123","sakila",330
6)
cursor = db.cursor()
sql = "UPDATE TEST_TABLE SET INCOME = INCOME + 500 WHERE SEX
= '%c'" % ('F')
try:
    cursor.execute(sql)
    db.commit()
    print("The records updated successfully!")
except:
    db.rollback()
    print("The records were not updated!")
db.close()
```

Мы создаем строку для запроса на обновление и запускаем ее с помощью

```
cursor.execute() .
```

Аналогично мы можем следующим образом удалить записи:

```
import pymysql
db =
pymysql.connect("localhost","testuser","test123","sakila",330
6)
cursor = db.cursor()
sql = "DELETE FROM TEST_TABLE"
try:
    cursor.execute(sql)
    db.commit()
```

```
    print("The records deleted successfully!")
except:
    db.rollback()
    print("The records were not deleted!")
db.close()
```

И мы можем даже следующим образом удалить таблицу:

```
import pymysql
db =
pymysql.connect("localhost", "testuser", "test123", "sakila", 330
6)
cursor = db.cursor()
sql = "DROP TABLE TEST_TABLE"
try:
    cursor.execute(sql)
    print("The table dropped successfully!")
except:
    print("The object does not exist!")
db.close()
```

MySQL и Pandas

Pandas — очень популярная библиотека обработки данных для Python 3. Мы можем установить ее, выполнив следующую команду в командной строке ОС:

```
pip3 install pandas
```

Pandas — очень большая тема, и ее сложно изучить в маленькой главе. Мы можем изучить библиотеку, посетив веб-страницу проекта, расположенную по URL-адресу <https://pandas.pydata.org>.

Давайте напишем простую программу на Python для чтения таблицы MySQL в структуру данных Pandas, известную как dataframe. Давайте рассмотрим очень простой пример:

```
import pymysql
db = pymysql.connect("localhost", "testuser", "test123",
"world", 3306)
import pandas as pd
df1 = pd.read_sql('select * from country', db)
print(df1)
```

В приведенной выше программе мы импортируем `pandas` и читаем данные из SQL-запроса с помощью функции `pd.read_sql()`. Прочитанные данные загружаются в кадр данных `Pandas`, который мы печатаем в командной строке Python 3.

Заключение

В этой главе мы научились запускать простые программы Python. Мы изучили и продемонстрировали взаимодействие между Python 3 и MySQL. Мы научились подключаться к схеме и выполнять различные операторы с помощью программ Python 3. Мы также увидели, как загружать записи из таблицы в фрейм данных `Pandas`. Мы можем использовать `MariaDB` вместо `MySQL`. Код Python 3 при этом останется таким же. Нам просто нужно внести изменения в параметр подключения для подключения к `MariaDB`.

Что следует помнить

- Мы можем запускать запросы `MySQL` из программы Python 3,
- Мы можем загрузить записи таблицы в фрейм данных `Pandas`.
- Нам нужно использовать библиотеку `pymysql` для подключения Python 3 к базе данных `MySQL` или `MariaDB`.

Вопросы с выбором правильного ответа

1. Какую библиотеку мы используем для подключения `MySQL/MariaDB` к Python 3?
 - a) `pymysql`
 - b) `pandas`
 - c) `numpy`
 - d) `scipy`
2. Можем ли мы выполнить `DML` в `MySQL/MariaDB`, используя Python 3?
 - a) Да
 - b) Нет

Ответы на вопросы

1. a)
2. a)

Вопросы

1. Объясните разницу между режимом сценария и интерактивными режимами Python 3?
2. Как мы можем загрузить записи таблицы экземпляра MySQL в фрейм данных Pandas?

Ключевые термины

Python 3, Pandas, MySQL, MariaDB, pymysql