СРЕДНЕЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

Серия основана в 2001 году

С.А. Мартишин, В.Л. Симонов,
М.В. Храпченко

БАЗЫ ДАННЫХ

Практическое применение СУБД SQL- и NoSQL-типа для проектирования информационных систем

УЧЕБНОЕ ПОСОБИЕ

Рекомендовано Учебно-методическим советом СПО в качестве учебного пособия для учебных заведений, реализующих программу среднего профессионального образования по специальностям УГС 09.02.00 «Информатика и вычислительная техника»

Москва
ИД «ФОРУМ» — ИНФРА-М
2019

УДК 004.6(075.32)
ББК 32.973я723
М29

Рецензенты:
доктор технических наук, профессор кафедры информационных систем, сетей и безопасности ФГБОУ ВО «Российский государственный социальный университет» *В.А. Сизов*;
доктор технических наук, доцент, директор военного института при ФГБОУ ВО «Московский авиационный институт (национальный исследовательский университет)» *В.И. Гончаренко*;
доктор педагогических наук, доктор социологических наук, профессор кафедры прикладной информатики ГАОУ ВО «Московский городской педагогический университет» *А.И. Каптерев*

Мартишин С.А.

М29 Базы данных. Практическое применение СУБД SQL- и NoSQL-типа для проектирования информационных систем : учеб. пособие / С.А. Мартишин, В.Л. Симонов, М.В. Храпченко. — М. : ИД «ФОРУМ» : ИНФРА-М, 2019. — 368 с. — (Среднее профессиональное образование).

ISBN 978-5-8199-0785-6 (ИД «ФОРУМ»)
ISBN 978-5-16-013889-3 (ИНФРА-М)

Учебное пособие предназначено для изучения практического применения СУБД SQL- и NoSQL-типа при проектировании информационных систем. Проведены аналогии между базами данных SQL- и NoSQL-типа. Полученные теоретические знания закрепляются при выполнении цикла лабораторных работ. Работа с SQL СУБД изучается на примере СУБД MariaDB. Рассмотрен вопрос использования хорошо зарекомендовавших себя в работе с MySQL приложений (например, phpMyAdmin, MySQL Workbench). Работа с NoSQL СУБД изучается на примере СУБД MongoDB с использованием оболочки Robomongo. Показана возможность использования универсального менеджера баз данных DBeaver для одновременной работы с СУБД MariaDB и MongoDB. Приведены примеры информационных систем на базе обоих типов СУБД — SQL и NoSQL.

Предназначено для студентов учреждений среднего профессионального образования, обучающихся по укрупненной группе специальностей 09.02.00 «Информатика и вычислительная техника», а также бакалавров, магистрантов, аспирантов и специалистов в области проектирования информационных систем и баз данных, а также для всех, кто интересуется проектированием информационных систем.

УДК 004.6(075.32)
ББК 32.973я723

ISBN 978-5-8199-0785-6 (ИД «ФОРУМ»)
ISBN 978-5-16-013889-3 (ИНФРА-М)

Предисловие

Для современного этапа развития информационных систем и технологий характерно несколько тенденций. Первая из них — быстрое уменьшение материальной ресурсоемкости современной продукции и такое же стремительное замещение материальных ресурсов информационными. Вторая тенденция — быстрая изменчивость всех составляющих информационных систем и технологий, включая аппаратное и программное обеспечения. Рассмотрим данную тенденцию более подробно.

В зависимости от достигнутых технологических успехов на каждом из этапов развития человечества ставится соответствующая общественная проблема/задача. При этом главенствующим здесь является развитие технологий — изобретение полупроводников, памяти, элементов и систем телекоммуникаций и пр. Кроме того, технологическое развитие ненадолго остается достоянием какой-то научной группы, института или даже страны. Конкуренция вынуждает всех участников быть на одинаковом уровне, активно добывая сведения друг у друга.

Еще одной важной тенденцией современного этапа является все более возрастающий объем данных, и эти данные не только научного плана (биологические, геофизические, медицинские, генетические и другие исследования), но и значительный объем данных, генерируемых широким кругом пользователей — информация о сделанных покупках, приобретениях, денежных перечислениях, а также смс, звонки, размещение сообщений и фото в социальных сетях и пр. Указанные данные зачастую имеют различную структурированность, и они должны храниться, обрабатываться и использоваться для генерации отчетов, в том числе в онлайн-режиме (например, для оптимизации продаж крупными торговыми компаниями или для выявления террористических угроз из обмена сообщениями в социальных сетях). Следовательно, именно громадный объем и необходимость онлайн-обра-

ботки информации различной структурированности — вот проблема современного этапа.

Нельзя не отметить следующее. В природе давно имеются и решены подобные задачи, и ученым и исследователям достаточно внимательно изучать жизнь и поведение живых существ и переносить свои наблюдения в лаборатории и разработки. Не случайно активно развивается наука «бионика» — от наблюдений за полетом птиц для совершенствования авиационных конструкций до организации жизни «общественных» насекомых, например муравьев, пчел и т. п., где так же, как у человека, имеются задачи организации многоуровневого уклада жизнедеятельности при соответствующем обмене большими объемами информации. Однако в природе такие сообщества характеризуются определенной статичностью в развитии, т. е. медленное движение по эволюционной лестнице сообществ живых существ не идет ни в какое сравнение с высокоскоростным развитием современной цивилизации, поэтому ученым остается внимательно наблюдать за жизнью таких сообществ, пытаться понять их, заимствовать алгоритмы и переносить (по возможности) на человеческую жизнь. В этом случае будет значительно быстрее достигнут желаемый успех в развитии человечества.

Проводя анализ прошедших этапов развития техники и технологий, можно отметить проблематичность прогнозов для отдаленной перспективы из-за некоей случайности в появлении технологических открытий. Тем не менее в краткосрочной перспективе очевидна необходимость обработки огромных объемов неодинаково структурированной информации, для чего разрабатываются новые и/или модернизируются существующие алгоритмы и системы обработки данных.

Введение

Проблема обработки больших объемов данных («big data») достаточно острая. Множество областей — научные исследования, бизнес-аналитика, социальные сети, крупные распределенные web-приложения и др. — требуют возможности работы с разнородной и зачастую с различно структурированной информацией. При этом должна обеспечиваться одновременная работа серверов под большой нагрузкой, поскольку обработка данных во многих областях не допускает задержки. В ряде распределенных проектов пользователи одновременно и генерируют, и потребляют информацию, поэтому требуется обеспечение масштабируемости, надежности и согласованности данных. При этом данные сами по себе могут и не иметь сложной структуры (например, данные социальных сетей — фото, подписи, комментарии, лайки, ссылки и некоторые другие), однако их объем и скорость обработки очень высоки.

Таким образом, имеют место следующие тенденции — значительное увеличение объема данных, подлежащих хранению и обработке; существенное возрастание количества пользователей информационных систем с одновременным ростом числа транзакций (включая мультмедийную информацию на мобильных устройствах); сложность структурирования (используя технологию систем управления реляционными базами данных) значительного объема хранимой информации.

Следовательно, требования, которым должны отвечать современные информационные системы, состоят в необходимости быстрой обработки информации в условиях работы в многопользовательском режиме.

Традиционные СУБД на основе реляционных таблиц (SQL-ориентированные СУБД), несмотря на большую популярность, не справляются с возрастающими требованиями. Поэтому в последние несколько лет были разработаны системы с иными подходами к обработке информации, лучше справляющиеся с поставленными задачами. Таким подходом является NoSQL («не только SQL»). NoSQL-системы

не используют реляционную модель данных, в них отсутствует жесткая структура, поэтому значительно лучше обеспечивается горизонтальная масштабируемость и требуемая производительность.

Однако наряду с достоинствами NoSQL-системам присущи недостатки, например, для них не гарантировано выполнения требований ACID (Atomicity (Атомарности), Consistency (Согласованности), Isolation (Изолированности), Durability (Долговечности)). Также не имеется аналогов команд BEGIN TRANSACTION, COMMIT и ROLLBACK, следовательно, проблематично их использование для финансовых систем.

NoSQL-системы (СУБД NoSQL) значительно проще с точки зрения модели данных (по сравнению с классической реляционной моделью). При этом NoSQL-системы могут основываться на различных моделях хранения данных. Наиболее распространенными в настоящее время являются: системы «ключ—значение» (Key—Value Stores), документно-ориентированные СУБД (Document Stores), Bigtable-подобные базы данных (Extensible Record Stores / Wide Column Stores / Column Families) и базы данных на основе графов.

Существует значительное количество разнообразных NoSQL-систем — порядка 150 единиц, что вызвано ориентированностью на различные области применения и использованием различных моделей данных. Однако здесь имеет место существенный недостаток — отсутствие единого стандарта требований к таким системам. Указанный стандарт позволил бы облегчить синхронизацию данных и дал бы возможность разработчикам использовать в одном проекте различные СУБД.

Отметим, что в направлении разработки стандартов ведется определенная деятельность. Так, в 2011 году компания Couchbase объявила о выпуске нового языка запросов — UnQL (Unstructured Data Query Language), предназначенного для работы с неструктурированными данными. Синтаксис языка UnQL аналогичен синтаксису языка SQL, например, поддерживаются такие команды, как SELECT, DELETE, INSERT и UPDATE.

Также имеет место тенденция сближения SQL- и NoSQL-подходов. При этом совершенствование реляционных СУБД ведется, в частности, в направлении поддержки нетрадиционных для них типов данных (XML-документов и XPath-запросов). Одновременно развиваются и СУБД NoSQL. Таким образом, для разработчика представляется логичным использование инструментария, более соответствующего поставленной задаче, будь то реляционная СУБД, СУБД NoSQL-типа или их совместное использование в одной информационной системе для хранения разных типов информации. Отметим, что указанное совместное использование стало возможным благодаря появлению универсального средства для работы с SQL и NoSQL базами данных — DBeaver (относится к категории свободного программного обеспечения — СПО).

Практически во всех курсах дисциплин, формирующих профессиональные компетенции выпускника и связанных с изучением информационных систем и баз данных (такие дисциплины, как «Базы данных», «Распределенные базы данных», «Проектирование информационных систем», «Управление данными», «Проектирование программного обеспечения АСОИиУ», «Сетевые операционные системы» и т. д.), необходимо получить представление об обоих подходах в построении информационных систем с использованием СУБД SQL и NoSQL-типа.

Таким образом, ***целью*** настоящего учебного пособия является обучение приемам работы как с СУБД SQL, так и NoSQL-типа, проведение аналогии между указанными двумя типами СУБД, изучение как теоретических положений, так и практических навыков путем выполнения конкретных лабораторных работ. Насколько известно авторам, такое учебное пособие на сегодняшний день отсутствует, и представленное учебное пособие в определенной мере восполняет данный пробел.

Задачи настоящего учебного пособия следующие:

- изучение широко распространенной СУБД SQL-типа MariaDB (в качестве примера работы с SQL-системами), являющейся ответвлением СУБД MySQL со свободным статусом. Для работы с MariaDB в пособии используются приложения, хорошо зарекомендовавшие себя в работе с MySQL, например phpMyAdmin, MySQL Workbench;
- освоение (в качестве примера работы с NoSQL-системами) приемов работы в СУБД MongoDB при помощи командной строки и оболочки Robomongo;
- изучение приемов работы с мультплатформенным универсальным менеджером баз данных DBeaver (работает с большим числом СУБД как SQL-, так и NoSQL-типа). DBeaver является инструментом, в значительной степени облегчающим проектирование, реализацию, модификацию и работу с различными СУБД.

Учебное пособие построено в виде цикла лабораторных работ, последовательно проводящих учащихся через тонкости проектирования баз данных для обоих подходов — SQL и NoSQL. Каждая глава снабжена заданиями для самостоятельной работы и списком контрольных вопросов.

В приложениях учебного пособия помещены сведения о целостности данных, описание семейства стандартов IDEF, спецификации BSON, селекторов запросов (Query Selectors).

Формируемые профессиональные компетенции (ожидаемые результаты обучения) студентов следующие. Студент должен:

знать:

- основные этапы построения баз данных различных типов — SQL и NoSQL — при проектировании информационных систем;
- компоненты программных комплексов и баз данных;
- технологию проектирования, производства и сопровождения баз данных в информационных системах;
- основы безопасной работы с СУБД (аутентификация, назначение прав доступа для пользователей, резервное копирование, восстановление системы после сбоев и пр.);

уметь:

- участвовать во всех фазах проектирования, разработки, изготовления и сопровождения баз данных для ИС;
- разрабатывать модели компонентов информационных систем, включая модели баз данных;
- осуществлять прямой и обратный инжиниринг;
- составлять структурированные запросы к информационным ресурсам локализованных и распределенных баз данных (создавать, обновлять и удалять документы в коллекции СУБД MongoDB; осуществлять выборку данных из коллекций и т. д.);
- создавать на базе выбранной СУБД ядро информационной системы и целостный программный продукт;

владеть:

- навыками по созданию программного средства с использованием базы данных;
- основами администрирования сервера MariaDB;
- навыками создания ER-моделей и работой с EER-диаграммами;
- основами администрирования СУБД MongoDB;
- навыками репликации и шардинга в СУБД MongoDB.

Часть I
ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ СУБД SQL-ТИПА НА ПРИМЕРЕ MARIADB

Введение. Общие сведения о СУБД MariaDB

В январе 2009 года вышел первый релиз системы управления базами данных (СУБД) **MariaDB**, получившей название в честь младшей дочери ведущего разработчика Ульфа Майкла Видениуса (Ulf Michael Widenius). Именно Видениус был одним из авторов первой версии СУБД MySQL и основателем компании MySQL AB. Позже в связи с политикой лицензирования компании Oracle, которая в настоящее время является владельцем MySQL, занялся разработкой совместимой с MySQL СУБД MariaDB — СУБД (под лицензией GPL), права на которую принадлежат компании Майкла Видениуса Monty Program Ab.

Команды, интерфейсы взаимодействия и API MariaDB совместимы с теми, которые используются в MySQL, т. е. библиотеки и приложения, которые работают с MySQL, также должны работать на MariaDB. Исходя из этого и из опасений, что Oracle сделает СУБД MySQL несвободным программным обеспечением, разработчики Fedora, начиная с 19 версии, заменили MySQL на MariaDB.

Кратко характеризуя MariaDB, отметим, что данная СУБД поддерживает работу с таким популярным клиентским приложением для удаленного администрирования баз данных, как **phpMyAdmin**, наиболее популярными фреймворками, такими как **Yii** и **Zend**, системами управления содержимым (CMS), например **WordPress** и **Plone**, и даже **Moodle** — системой управления электронным обучением.

Также заметим, что многие популярные версии ОС Linux **Fedora** (начиная с 19 версии), а также **Gentoo**, **openSUSE**, **Red Hat Enterprise Linux** поддерживают работу с MariaDB.

Лабораторная работа 1
УСТАНОВКА СУБД MARIADB И ОСВОЕНИЕ РАБОЧЕГО ПРОСТРАНСТВА MYSQL WORKBENCH ДЛЯ РАБОТЫ С СУБД MARIADB

Цель: изучение принципов установки и работы с СУБД MariaDB.

Установка MariaDB

Дистрибутивы MariaDB для операционных систем (ОС) Linux, Solaris и Windows доступны на **downloads.mariadb.org**. Кроме того, MariaDB входит в состав дистрибутива FreeBSD, некоторых дистрибутивов на базе MAC ОС и наиболее распространенных ОС Linux: Ubuntu, openSUSE, ALTLinux, Fedora, Red Hat Enterprise Linux, Debian, Gentoo и др.

В настоящем учебном пособии в качестве операционной системы используется ОС Fedora (не ниже версии 20), как одна из наиболее простых в установке и подходящих для учебных целей операционных систем семейства Linux. В данный дистрибутив входит MariaDB, поэтому производим установку, набрав в командной строке следующее:

```
# yum install mariadb mariadb-server
```

Далее в терминале набираем команды (первая из которых запускает сервер, вторая дает возможность автоматического запуска при загрузке ОС):

```
# systemctl start mariadb.service
# systemctl enable mariadb.service
```

Изменяем установки межсетевого экрана (для стандартной графической оболочки Gnome **Обзор → Разное → Межсетевой экран**) — создать порт **«3306 tcp»** (рис. 1.1). Также необходимо в настройках

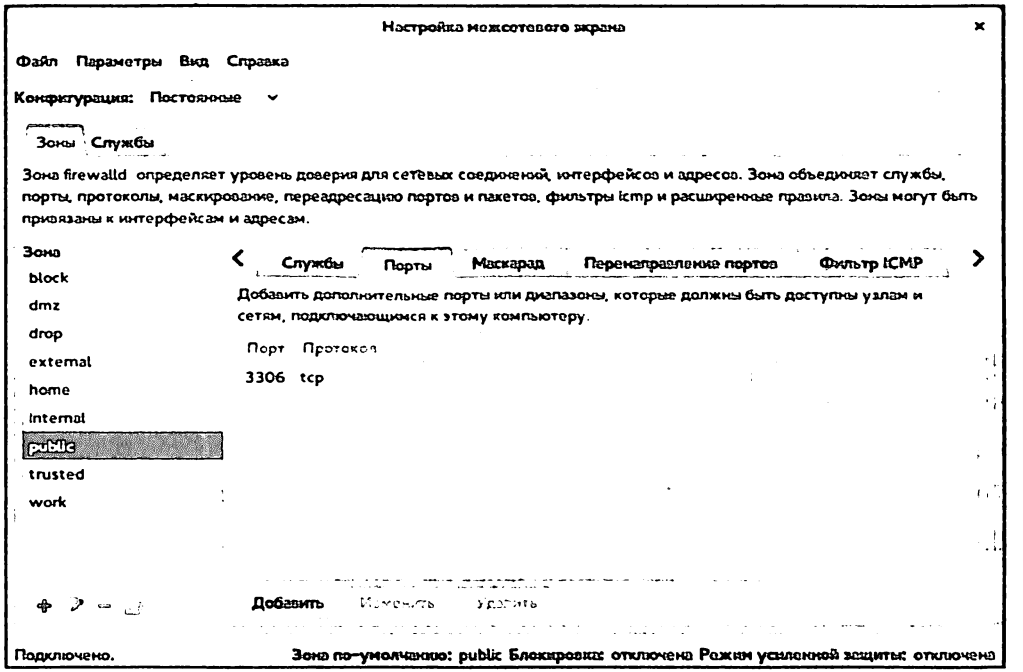


Рис. 1.1. Настройка межсетевого экрана

конфигурации выбрать опцию «Постоянные», поскольку динамические конфигурации работают только до перезагрузки.

Введем пароль **root** для управления MariaDB:

```
# mysqladmin -u root password пароль_администратора_MariaDB
```

Затем войдем в терминал под root:

```
# mysql -u root -p
```

Введем пароль администратора **MariaDB** и войдем в терминал **MariaDB** (рис. 1.2):

```
MariaDB [(none)]>
```

Для выхода из терминала MariaDB ввести **\q** и нажать клавишу **«Enter»**.

Далее для удобства работы будем использовать графический интерфейс. Как было сказано выше, для работы MariaDB можно использовать значительную часть приложений, хорошо зарекомендовавших себя с MySQL, например **phpMyAdmin** или **MySQL Workbench**.

```
root@milonga:~
Файл  Правка  Вид  Поиск  Терминал  Справка
  Проверка     : 1:mariadb-server-5.5.37-1.fc20.x86_64                  1/1

Установлено:
  mariadb-server.x86_64 1:5.5.37-1.fc20

Выполнено!
[root@milonga ~]# systemctl start mariadb.service
[root@milonga ~]# systemctl enable mariadb.service
ln -s '/usr/lib/systemd/system/mariadb.service' '/etc/systemd/system/multi-user.
target.wants/mariadb.service'
[root@milonga ~]# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2
Server version: 5.5.37-MariaDB MariaDB Server

Copyright (c) 2000, 2014, Oracle, Monty Program Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Рис. 1.2. Установка MariaDB

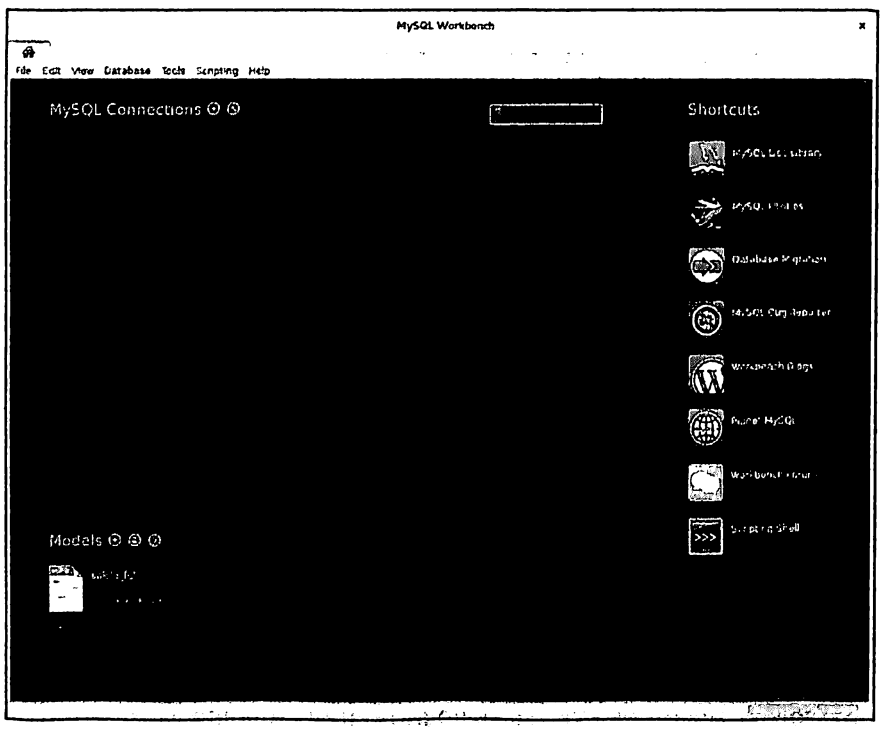


Рис. 1.3. Окно среды разработки MySQL Workbench

Для установки MySQL Workbench скачайте бесплатную версию с **http://dev.mysql.com/downloads/workbench** и установите ее. Вид экрана при вызове MySQL Workbench представлен на рис. 1.3. Затем произведем настройки (**Обзор → Показать Приложения → MySQL Workbench**). Для этого необходимо нажать на знак «+» (рис. 1.4) и произвести настройки, как показано на рис. 1.5:

Hostname: localhost (или 127.0.0.1).
Port: 3306.
Username: root.
Password: пароль_администратора_MariaDB.

Рис. 1.4. Переход в настройки MySQL Workbench

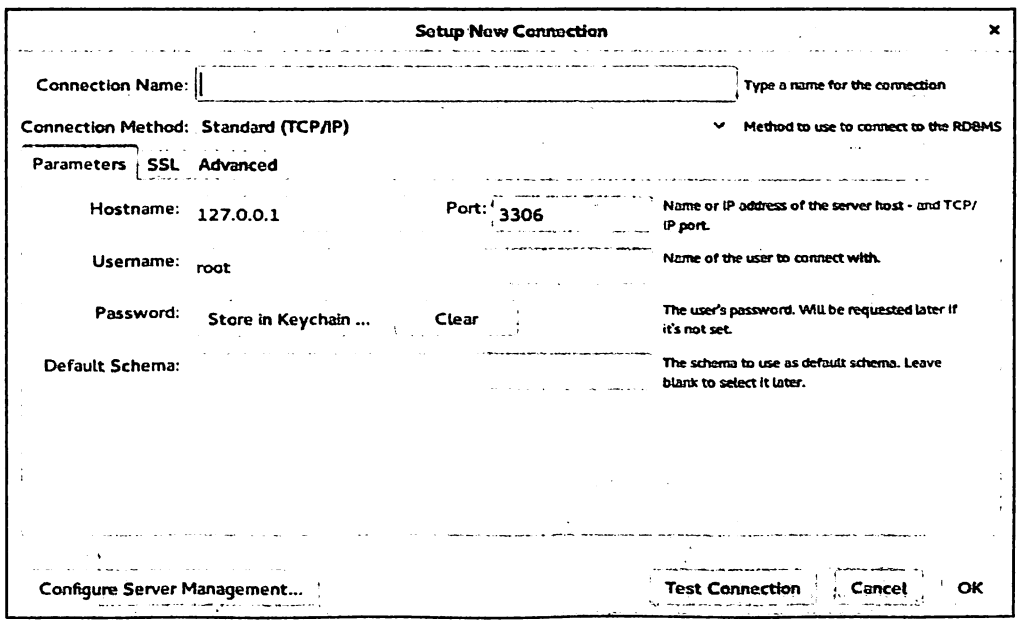


Рис. 1.5. Настройки создаваемого соединения в MySQL Workbench

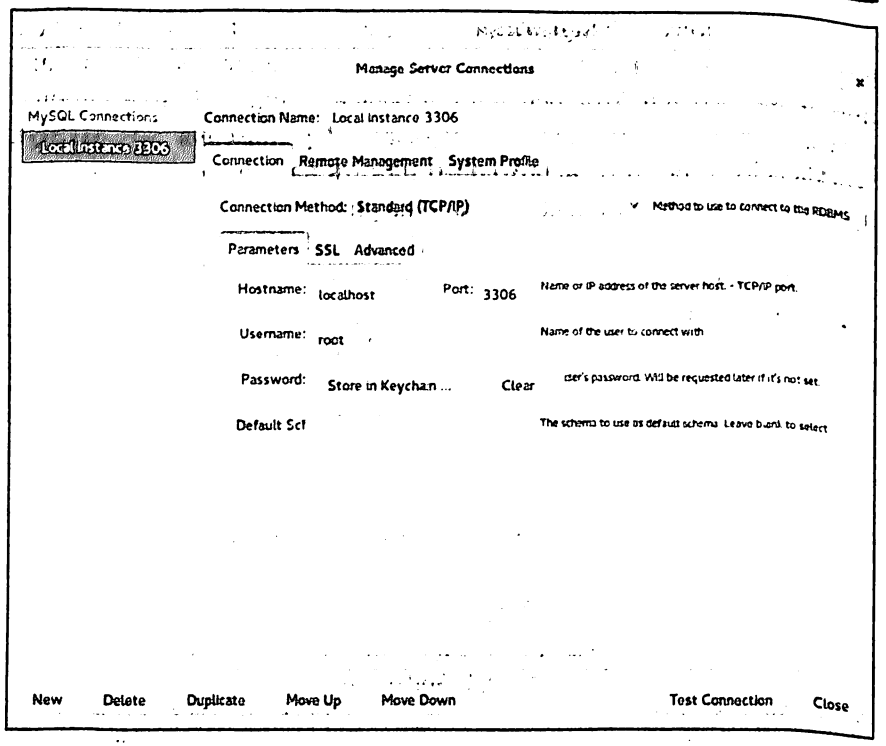


Рис. 1.6. Настройки соединения в MySQL Workbench

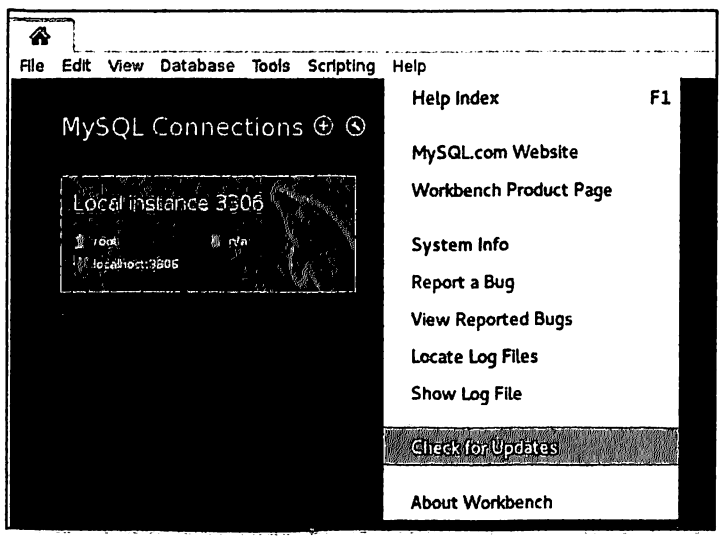


Рис. 1.7. Обновление MySQL Workbench

После нажатия на кнопку **OK**, если все шаги были проделаны аккуратно, то вы сможете работать со средой разработки MySQL Workbench.

В случае необходимости внесения изменений в параметры созданного соединения можно нажать на значок и произвести требуемые настройки (рис. 1.6).

Заметим, что из соображений безопасности желательно завести пользователя, отличного от **root**, т. е. не обладающего правами администратора (администрирование, создание аккаунтов и назначение привилегий рассмотрено ниже или см., например, [1]).

Если ранее на компьютере была установлена какая-либо предыдущая версия MySQL Workbench, то ее необходимо обновить. Для этого в пункте меню **Help** следует выбрать пункт **Check for Updates** и установить последнюю версию (рис. 1.7).

Создание простого примера базы данных в СУБД MariaDB в MySQL Workbench

После того как СУБД **MySQL** и **Workbench** были установлены и настроены, можно приступать к созданию базы данных [1]. При запуске открывается окно среды разработки **MySQL Workbench** (см. рис. 1.3).

На рис. 1.3 видно, что в верхней части окно содержит **основное меню** для работы с **файлами** (**File**), предусматривающее создание новой модели, открытие существующей, печать, выход и т. п., **редактирования** (**Edit**), предусматривающее отмену действия, копирование, вставку, вырезание, поиск. Эти пункты меню имеют практически тот же состав функций, что и хорошо известные программные продукты. Пункт меню **вид** (**View**) предусматривает настройку отображения, в том числе его увеличение и перемещение по таблицам.

Для операций с **базой данных** (выбора и подключения, прямого и обратного инжиниринга) используется пункт меню **Database**.

Пункт меню **инструменты** (**Tools**) позволяет работать с различными утилитами.

Пункт меню **скрипт** (**Scripting**) предназначен для создания нового или загрузки уже существующего скрипта из файла.

Пункт меню **помощь** (**Help**) содержит справочную информацию.

Окно разработки имеет три области: в левом верхнем углу различные соединения с базами данных, внизу слева кнопки для работы со

средствами моделирования данных и справа — утилиты, в том числе и для администрирования.

Если войти в пункт меню **Help** и выбрать пункт **About Workbench**, то можно узнать версию продукта. Дальнейшее изложение будет производиться для версии 10.0.21 (на момент написания книги использовалась последняя версия). Более ранние версии могут отличаться от рассматриваемой пунктами меню и видом экрана. Тем не менее различия не являются значительными, и освоение любой версии позволяет достаточно легко перейти к использованию другой.

Справа находится панель **Shortcuts**, при помощи которой можно получить доступ к различной справочной информации через Интернет.

Для создания базы данных необходимо установить новое соединение или выбрать уже имеющееся. Для выбора имеющегося щелкаем мышью по выбранному соединению (рис. 1.8, соединение **localhost**).

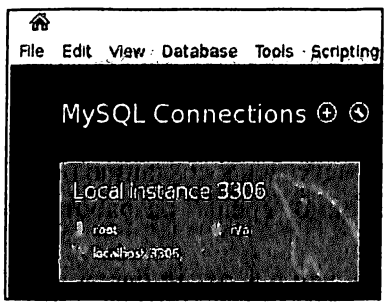


Рис. 1.8. Выбор текущего соединения

Для создания нового соединения следует выбрать пункт меню **Database → Connect to Database** (рис. 1.9), перейти в диалог управления соединением с базой данных (**Manager Server Connection**) и выбрать из раскрывающегося меню **localhost**. В качестве протокола выбирается стандартный протокол соединения TCP/IP. Остальные параметры (вкладки **SSL** и **Advanced**) настраивать рекомендуется только в том случае, если пользователь понимает последствия своих действий. В противном случае остальные настройки необходимо оставить по умолчанию.

Далее появится окно для ввода пароля (рис. 1.10), после ввода которого и нажатия кнопки OK откроется вкладка редактора **SQL** (рис. 1.11), где можно создавать, модифицировать и удалять как саму базу данных, так и ее содержимое.

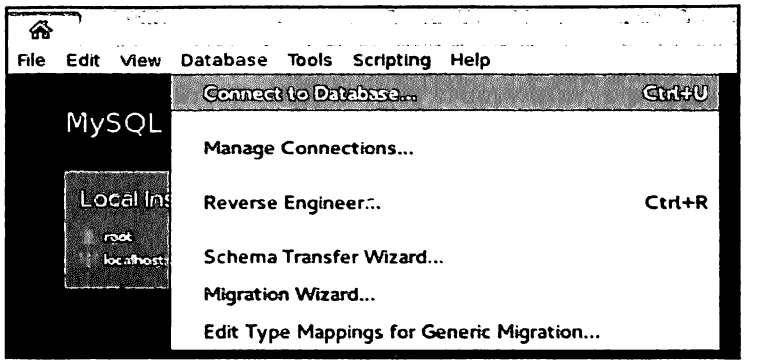


Рис. 1.9. Соединение с базой данных

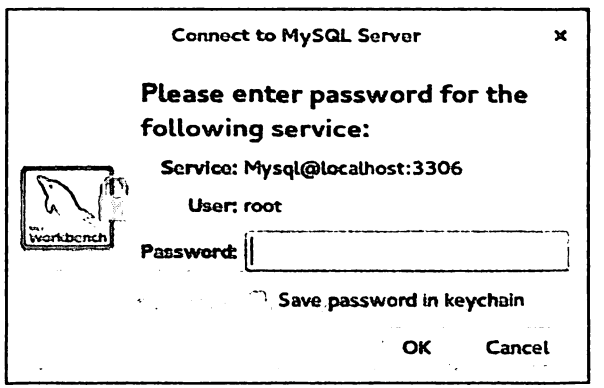


Рис. 1.10. Ввод пароля

Пользовательский интерфейс SQL-редактора содержит следующие основные элементы:

- основное меню в верхней части окна (см. рис. 1.11);
- панель управления (рис. 1.12);
- панель управления окна запросов (см. ниже рис. 1.18);
- навигатор боковой панели, включает разделы: Management, Instance, Performance и Schema views (см. рис. 1.11);
- вверху справа имеются кнопки для отключения Навигатора (левая), правая для отключения правой панели SQL Additions и нижняя для отключения вывода (Output).

Основное меню окна редактора в верхней части (см. рис. 1.11) предназначено для работы с файлами, базами данных, скриптами. Обратите внимание, что вверху указано текущее соединение. В данном меню имеются два пункта, которые мы будем использовать в основ-

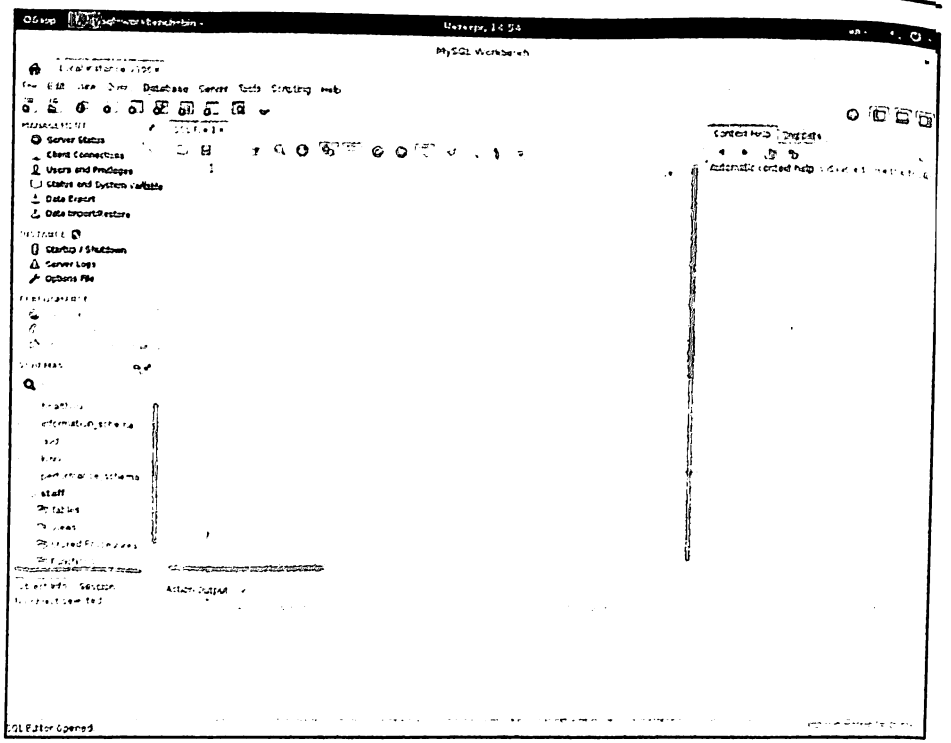


Рис. 1.11. Вкладка редактора SQL

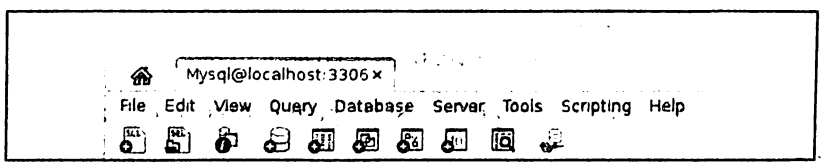


Рис. 1.12. Элементы панели управления SQL-редактора

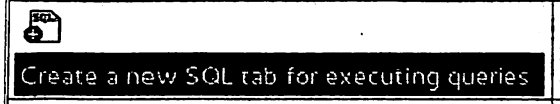
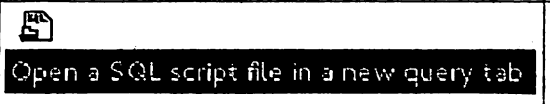
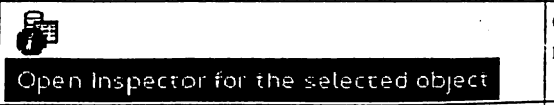
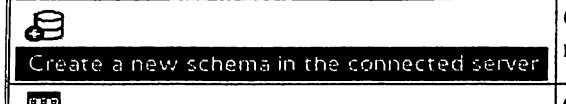
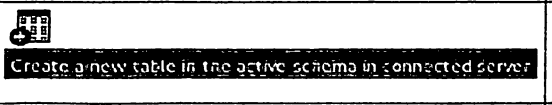
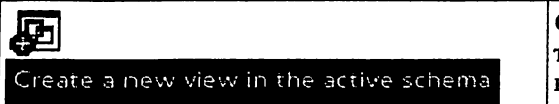
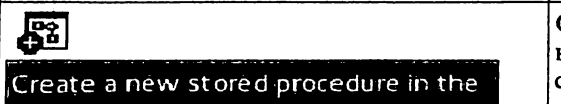
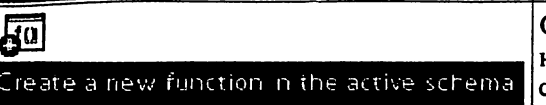
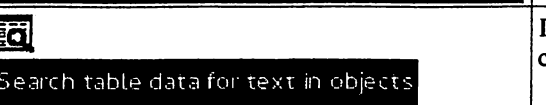
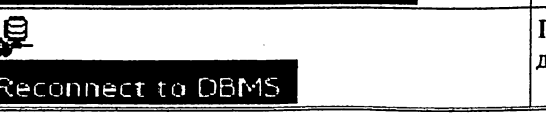
ном меню **Workbench: запрос (Query)**, который позволяет осуществить запуск запросов и **сервер (Server)**, который предназначен для администрирования сервера СУБД (заметим, что большая часть пунктов меню **Server** совпадает с пунктами меню, расположенными слева в разделе Management).

Заметим также, что вкладка основного окна может называться **Query** или **SQL File** в зависимости от способа соединения. Все элементы управления для вкладок совпадают. Различие связано с настройками **Workbench** (более подробную информацию можно найти на сайте **http://dev.mysql.com/doc/workbench/en/index.html**).

Под пунктами основного меню расположены пиктограммы панели управления SQL-редактора (см. рис. 1.12).

Слева направо пиктограммы имеют следующие значения (табл. 1.1).

Таблица 1.1. **Элементы панели управления SQL-редактора**

Элемент управления	Описание
	Создание нового окна для ввода кода исполняемого запроса (SQL-скрипта)
	Открыть ранее сохраненный SQL-скрипт, подготовленный для выполнения. Скрипт будет отображен в области **SQL File**
	Открывает окно Инспектора для выбранного объекта
	Создать новую базу данных на подключенном сервере
	Создать новую таблицу в активной базе данных на подключенном сервере
	Создать новое представление в активной базе данных на подключенном сервере
	Создать новую процедуру в активной базе данных на подключенном сервере
	Создать новую функцию в активной базе данных на подключенном сервере
	Поиск данных в таблицах, соответствующих заданному шаблону
	Переподключиться к серверу баз данных

Если выбрать первый пункт меню (табл. 1.1) «Создание нового окна для ввода кода исполняемого запроса (SQL-скрипта)», то будет открыта новая вкладка для работы с запросами. Подробнее окно запросов будет рассмотрено ниже.

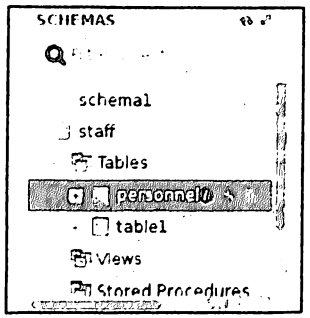


Рис. 1.13. Выбор объекта

Пункт меню «Открыть ранее сохраненный SQL-скрипт, подготовленный для выполнения» позволит выбрать сохраненный файл и открыть SQL-скрипт.

Для получения данных о выбранном объекте (следующий элемент управления) необходимо слева в окне (рис. 1.13) выбрать объект: базу данных, таблицу базы данных

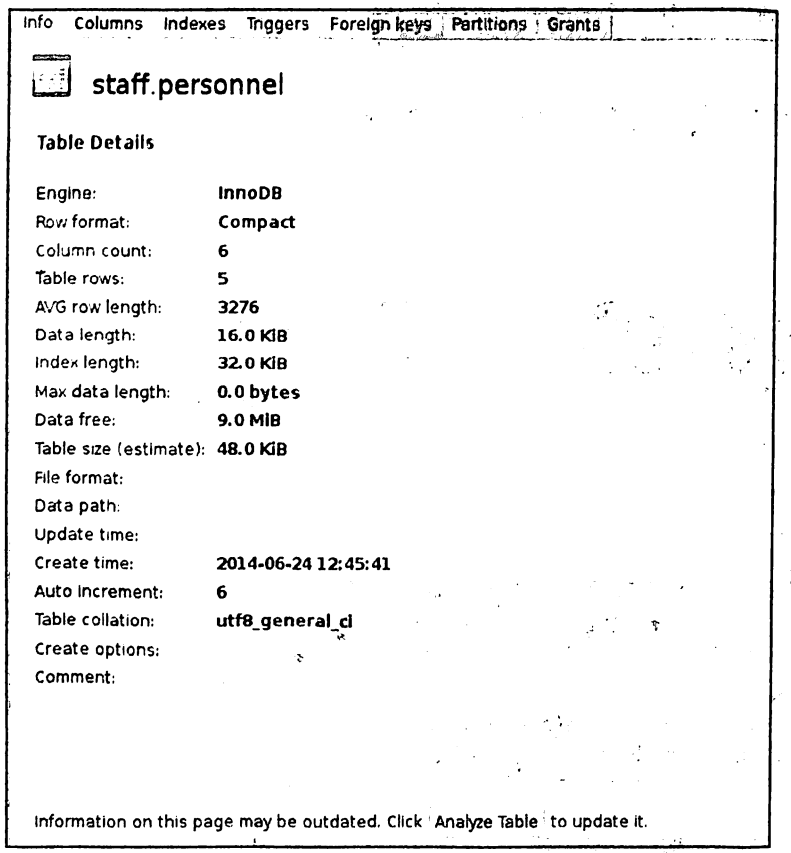


Рис. 1.14. Данные о выбранном объекте

и пр., и при нажатии на значок данные об объекте будут открыты в основном окне. Например, на рис. 1.14 показана информация, которая будет представлена в основном окне для таблицы **personnel** базы данных **staff**.

Обратите внимание, что рядом находятся еще два значка: для получения информации о структуре выбранной таблицы базы данных (может быть использовано для перехода к работе с выбранной таблицей) (рис. 1.15) и значок для отображения данных, внесенных в таблицу (рис. 1.16).

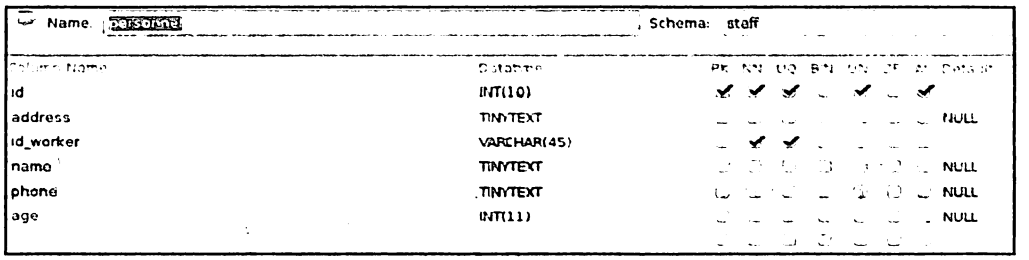


Рис. 1.15. Структура выбранной таблицы

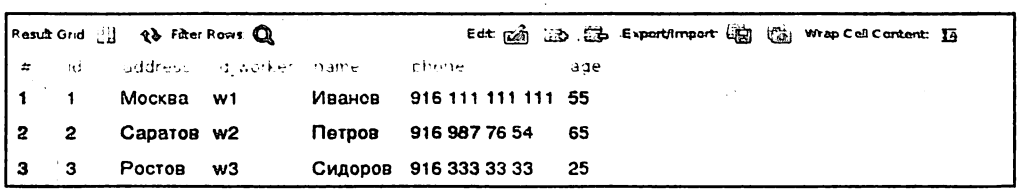


Рис. 1.16. Работа с данными выбранной таблицы

Выбор пиктограммы (см. рис. 1.12), т. е. пункта меню «Создать новую базу данных на подключенном сервере», позволяет создать новую базу данных, задать ее имя, выбрать кодировку (рис. 1.17). Создать новую таблицу (**Create a new table...**) в активной базе данных на подключенном сервере можно, выбрав соответствующий элемент управления . Подробнее процесс создания базы данных рассмотрен ниже.

Следующий элемент управления (**Create a new view...**) дает возможность создать новое представление в активной базе данных на подключенном сервере в открывающейся вкладке, которое удобно использовать при работе со сложными и большими, а также перекрестными запросами.

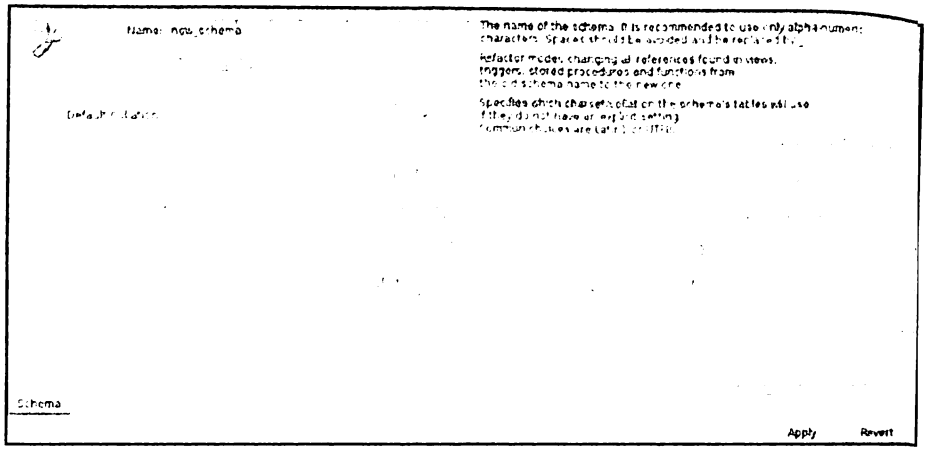


Рис. 1.17. Создание новой базы данных

Далее, при вызове пункта меню «Создать новую процедуру в активной базе данных на подключенном сервере» или «Создать новую функцию в активной базе данных на подключенном сервере», открывается соответствующее окно для ввода кода. Функция от процедуры отличается тем, что первая возвращает некоторое значение. Ниже приведены фрагменты кода, который в качестве шаблона задан на соответствующей странице:

```
CREATE PROCEDURE `new_procedure` ()
BEGIN
.....................
END
```

или

```
CREATE FUNCTION `new_function` ()
RETURNS INTEGER
BEGIN
.....................................
RETURN 1;
END
```

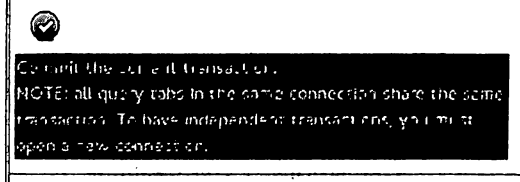
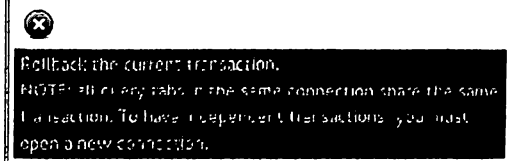
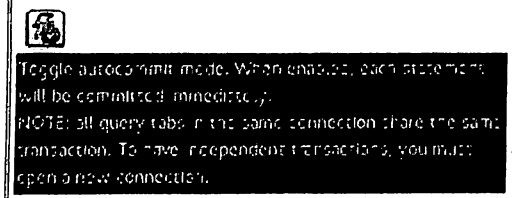
Далее рассмотрим пиктограммы панели управления окна запросов (рис. 1.18). Заметим, что в главном меню пункты **File**, **Edit** и **Query** содержат все возможные действия, связанные с редактированием и обработками запросов. Наиболее часто вызываемые команды с соответствующими пиктограммами вынесены на панель управления SQL-редактора, а их значения приведены ниже (табл. 1.2).

Рис. 1.18. Пиктограммы панели управления окна запросов

Таблица 1.2. **Пиктограммы панели управления окна запросов**

Элемент управления	Описание
Open a script file in this editor	Открыть файл, содержащий скрипт в данном редакторе
Save the script to a file.	Сохранить в файле, указанном пользователем SQL-скрипта
Execute the selected portion of the script or everything, if there is no selection	Запуск на выполнение части или всего SQL-скрипта. Результат будет выведен в одной или нескольких результирующих таблицах
Execute the statement under the keyboard cursor	Запуск на выполнение SQL-оператора, на который указывает курсор. Результат будет выведен в одной или нескольких результирующих таблицах
Execute the EXPLAIN command on the statement under the cursor	Выполнение команд EXPLAIN, на которые указывает курсор (см. **EXPLAIN** Syntax), показывает дополнительную информацию о выполнении SQL-запроса
Stop the query being executed (the connection to the DB server will not be restarted and any open transactions will remain open)	Прекращение выполнения текущего SQL-скрипта. Эта команда не перезапускает соединение с сервером базы данных
Toggle whether execution of SQL script should continue after failed statements	Если кнопка нажата, то выполнение скрипта остановится на команде, которая содержит ошибку. Если кнопка не нажата, то скрипт будет выполняться (в том числе и неверный код), возможно, с генерацией дополнительного результирующего сообщения

Окончание табл. 1.2

Элемент управления	Описание
	Зафиксировать транзакцию
	Откат транзакции. Замечание: все вкладки запросов в одном соединении относятся к одной транзакции. Для независимых транзакций следует устанавливать разные соединения
	Если переключатель нажат, транзакции подтверждаются автоматически. Это относится ко всем вкладкам в одном соединении. Для независимых транзакций следует устанавливать разные соединения
	Сохранить текущий оператор или запрос как фрагмент скрипта
	Переформатировать SQL-скрипт
	Показывать панель поиска для редактора
	Если переключатель нажат, то отображаются невидимые спецсимволы: пробелы, переход на новую строку и т. п.
	Если переключатель нажат, то осуществляется перенос на другую строку длинных строк

Стандартная конфигурация панели управления окна запросов следующая: нажаты кнопки остановки выполнения скрипта на команде, которая содержит ошибку, ограничения числа строк в возвращаемом запросе и автоматического подтверждения транзакций.

Заметим, что если переключатель **Autocommit** не нажат, то на панели кнопки **Commit** и **Rollback** будут доступны для нажатия. Если переключатель нажат, то транзакция принимается автоматически. Переключение доступно также через меню **Query** (рис. 1.19).

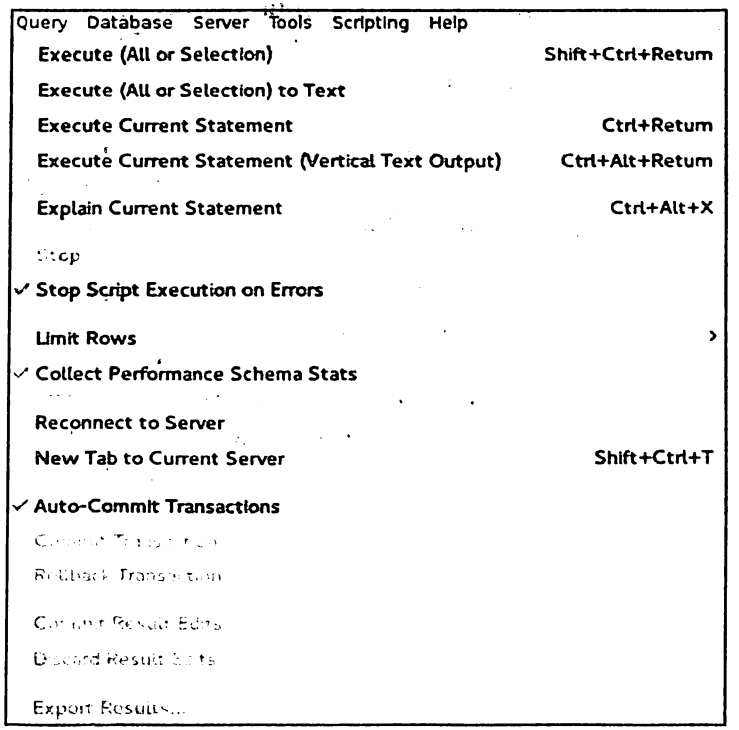


Рис. 1.19. Меню **Query**

Навигатор боковой панели расположен в основном окне слева (см. рис. 1.13, 1.20) и включает возможности для **Управления (Management)** соединением и получением связанных с ним данных, например о состоянии сервера, пользователях и привилегиях и пр. **Экземпляр (Instance)**: опции для взаимодействия с соединением с СУБД, такие как **Start / Stop**, просмотр журнальных файлов и файл настройки опций, **Производительность (Performance)**: обеспечивает графическое

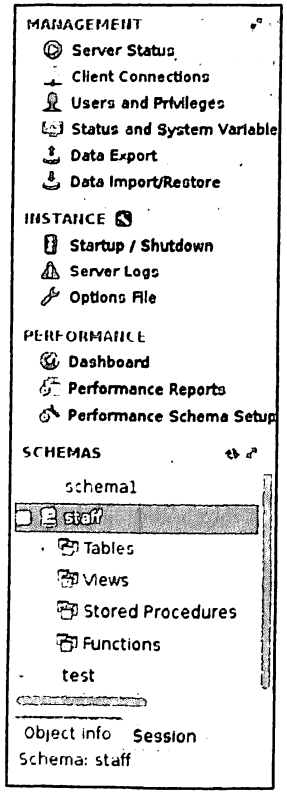
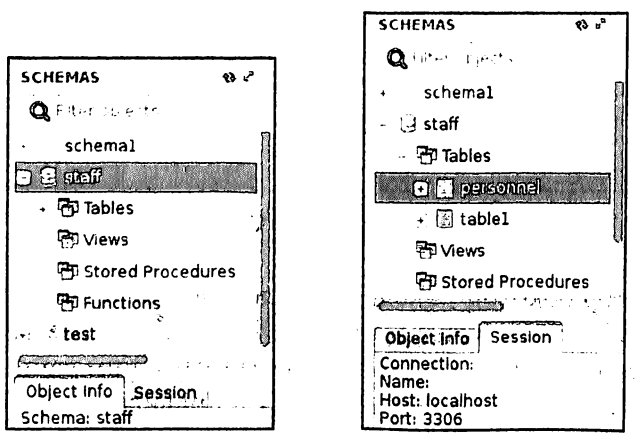


Рис. 1.20. Навигатор боковой панели

представление ключевых статистических данных из состояния сервера, **Схемы (Schemas)**: сведения о базах данных.

Ниже расположена панель **Информация**, содержащая две вкладки. Информация об объекте (**Object Info**), в которой отображается информация о выбранном объекте схемы. Например, отображается структура столбцов в случае, если выбрана таблица или определение функции для функции (рис. 1.21, *а*). Вкладка сессии (**Session**) обобщает информацию об объекте, например имя подключения, логин пользователя и порт (рис. 1.21, *б*).

В окне редактора справа на боковой панели находятся две вкладки **Context Help** для помощи и **Snippets** для хранения фрагментов SQL-скриптов (рис. 1.22). Это могут быть встроенные или пользовательские фрагменты, которым можно давать имена (при сохранении фрагменту необходимо давать имя). Эти фрагменты могут быть просмотрены и отредактированы на боковой панели фрагментов. Чтобы загрузить фрагмент в области SQL-запросов либо выбрать фрагменты, необходимо



а *б*

Рис. 1.21. *а* — информация об объекте; *б* — информация о сессии

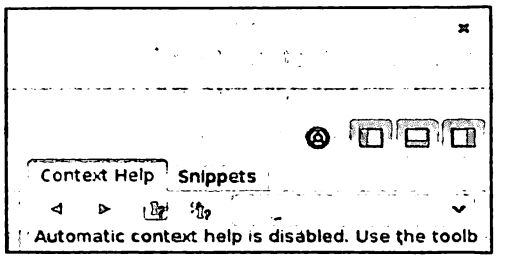
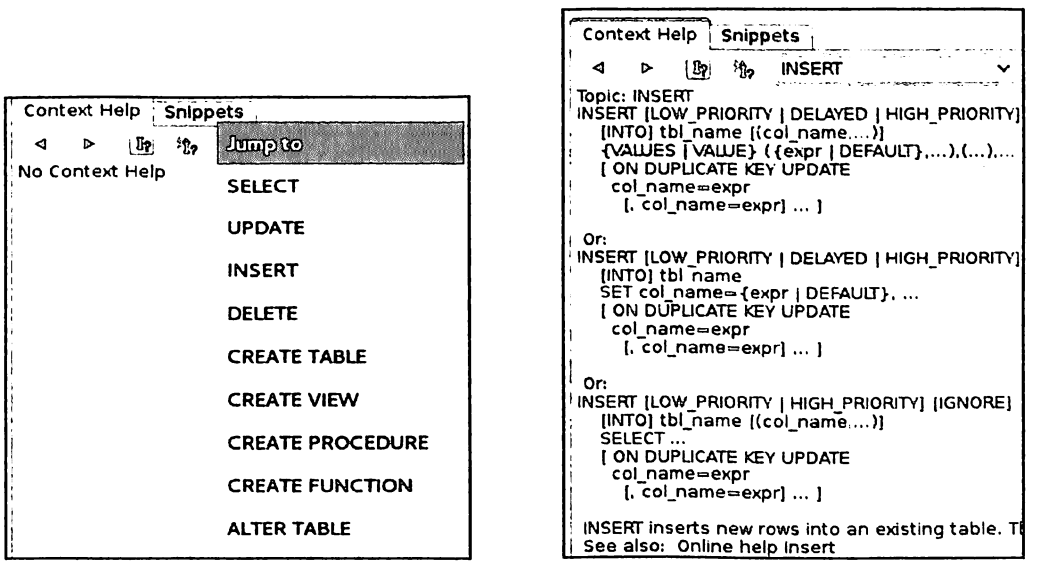


Рис. 1.22. Вкладки **Context Help** и **Snippets**

щелкнуть правой кнопкой мыши на нужный фрагмент и нажать «Вставить».

Первая вкладка **Context Help** предназначена для получения информации об операторах языка SQL. В раскрывающемся списке (рис. 1.23, *а*) выбирается оператор и описываются способы его применения (рис. 1.23, *б*).

Кнопки окна контекстного поиска имеют следующие названия: **One topic back** (вернуться на одну тему назад), **One topic forward** (пе-



а *б*

Рис. 1.23. Закрытие окна **Навигатора**:
а — раскрывающийся список; *б* — выбор оператора

рейти к следующей теме), **Toggle automatic context help** (включить режим контекстной помощи), **Get context help for the item at the current caret position** (получить контекстную помощь по слову, на которое указывает курсор). Функционирование этих кнопок полностью соответствует их названию.

Вкладка **Snippets** для пользовательских фрагментов скриптов и встроенных опций под названием **DB Mgmt (Database Management)**, **SQL DDL (SQL Data Definition Language)** и **SQL DML (SQL Data Manipulation Language)** (рис. 1.24). Фрагментам можно дать названия, их можно просматривать и редактировать во вкладке **Snippets**.

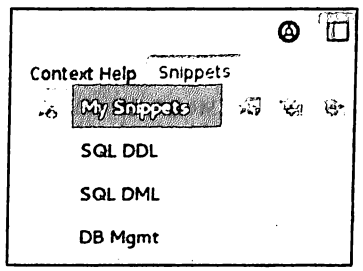


Рис. 1.24. Вкладка **Snippets**

Также вверху справа на боковой панели (см. рис. 1.22) расположены три кнопки, с помощью которых можно менять пространство окна редактора, и

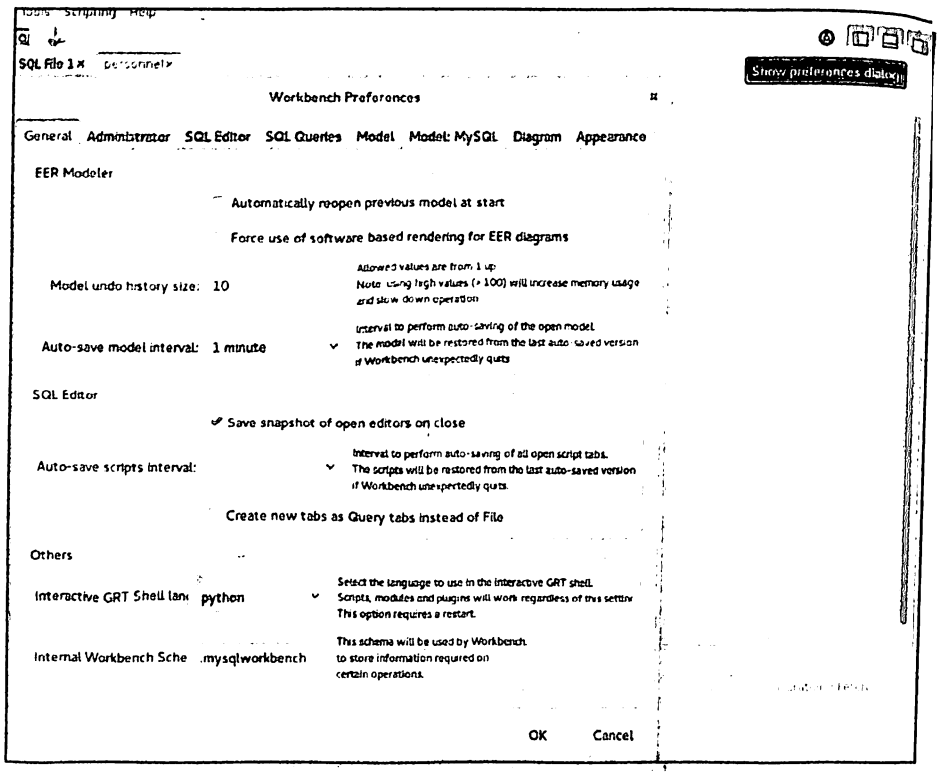


Рис. 1.25. Закрытие окна **Навигатора**

кнопка **Show preference dialog** (рис. 1.25). Вызов диалога предпочтений позволяет произвести настройки для конкретной базы данных и подробнее будет рассмотрен ниже.

Кнопки, с помощью которых можно менять пространство окна редактора, позволяют убрать/восстановить отображаемые области. Например, на рис. 1.26 показано, что при нажатии на правую кнопку изменения конфигурации окна редактора закроется вкладка **Context Help** и **Snippets**. Если нажать на левую кнопку, закроется окно Навигатора. Центральная кнопка закроет вкладку, расположенную внизу окна редактора.

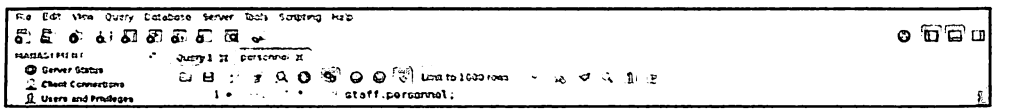


Рис. 1.26. Закрытие вкладок **Context Help** и **Snippets**

Перейдем к созданию элементарной таблицы в базе данных. Создадим базу данных, выбрав пиктограмму «Создать новую базу данных на подключенном сервере» (см. рис. 1.12). Пусть база данных называется **mydb1**. Затем в раскрывающемся списке выберем тип кодировки данных. Заметим сразу, что в стандартных случаях следует выбирать кодировку UTF-8 (utf-8 — utf8_general_ci) (рис. 1.27). Далее

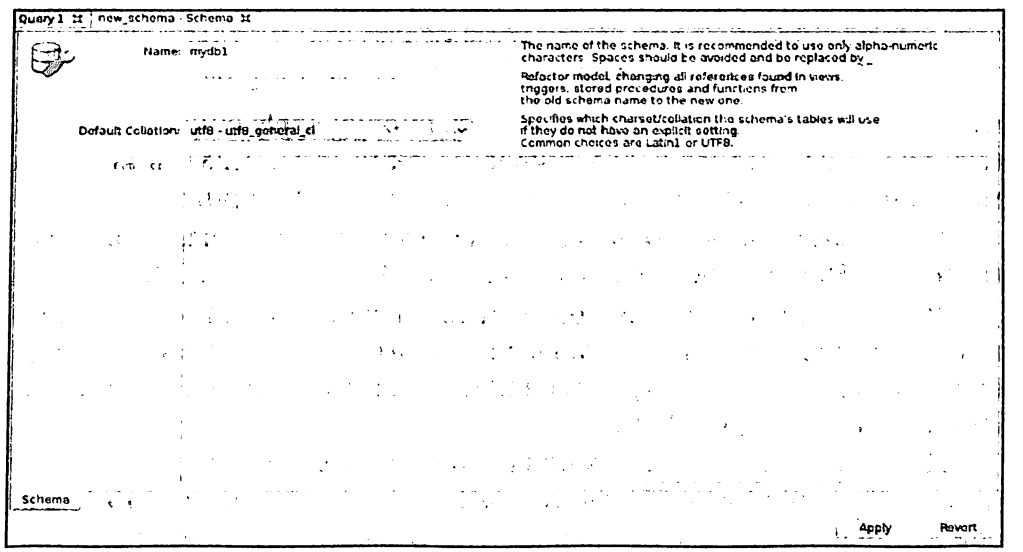


Рис. 1.27. Выбор кодировки базы данных

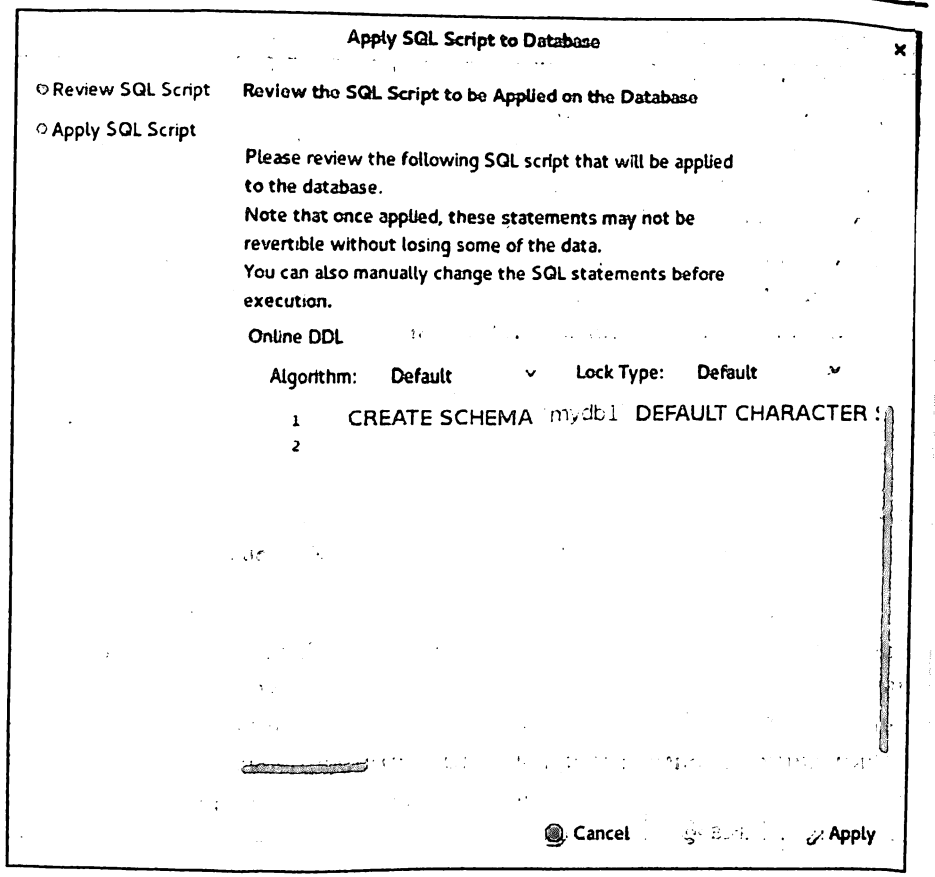


Рис. 1.28. Просмотр скрипта создания базы данных

для подтверждения необходимо нажать кнопку **Apply** внизу страницы (см. рис. 1.27). После этого откроется окно (рис. 1.28), в котором можно осуществлять просмотр и исправление SQL-скриптов.

Когда база данных создана, она появляется в левой части окна редактора — Навигаторе, в разделе **Схемы (Schemas)** (рис. 1.29).

У базы данных имеется контекстное меню, позволяющее выполнять различные команды: устанавливать базу данных в качестве текущей, отправлять SQL-скрипты в SQL-редактор, изменять, удалять, обновлять и т. п. (рис. 1.30).

Для создания таблиц можно воспользоваться контекстным меню (рис. 1.31) или вызвать команду нажатием на соответствующий значок на панели управления рис. 1.12. Выберем пункт **«Создать таблицу» (Create Table)**.

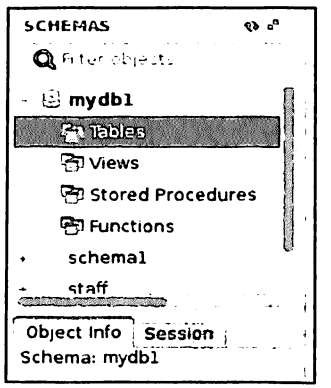


Рис. 1.29. Созданная база данных отображена в Навигаторе

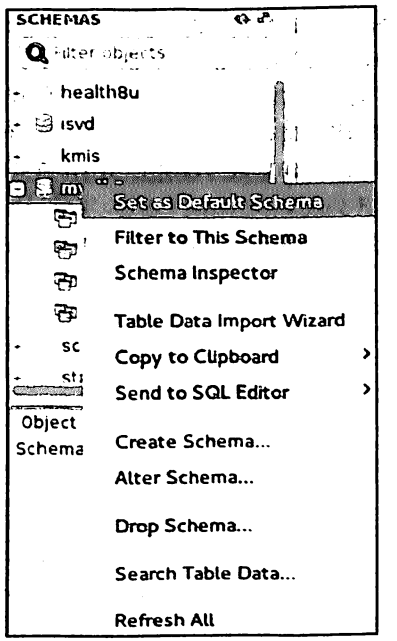


Рис. 1.30. Контекстное меню базы данных

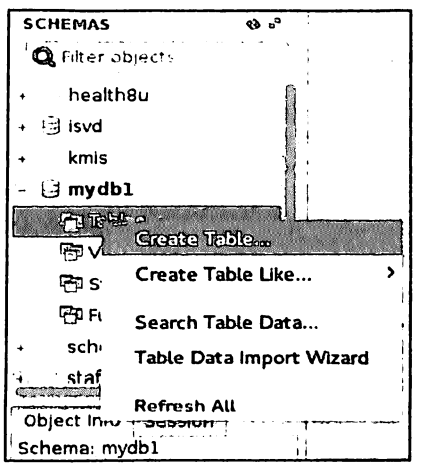


Рис. 1.31. Создание новой таблицы в базе данных

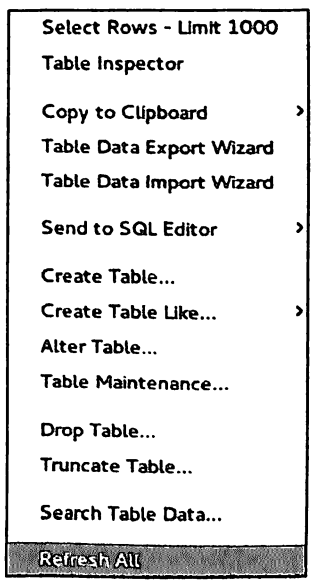


Рис. 1.32. Контекстное меню созданной таблицы в базе данных

Когда таблица создана, контекстное меню для работы с ней имеет вид, как показано на рис. 1.32. Таблицу можно редактировать, копировать, изменять, удалять, обновлять.

Например, если необходимо удалить таблицу с именем **table1**, выбирается соответствующий пункт меню **Drop Table** и на экране появляется сообщение (рис. 1.33) об удалении таблицы, которое можно принять (**Apply**) или отклонить (**Cancel**).

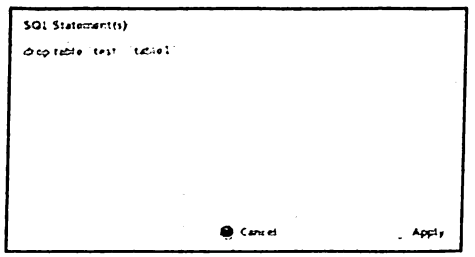


Рис. 1.33. Сообщение об удалении таблицы

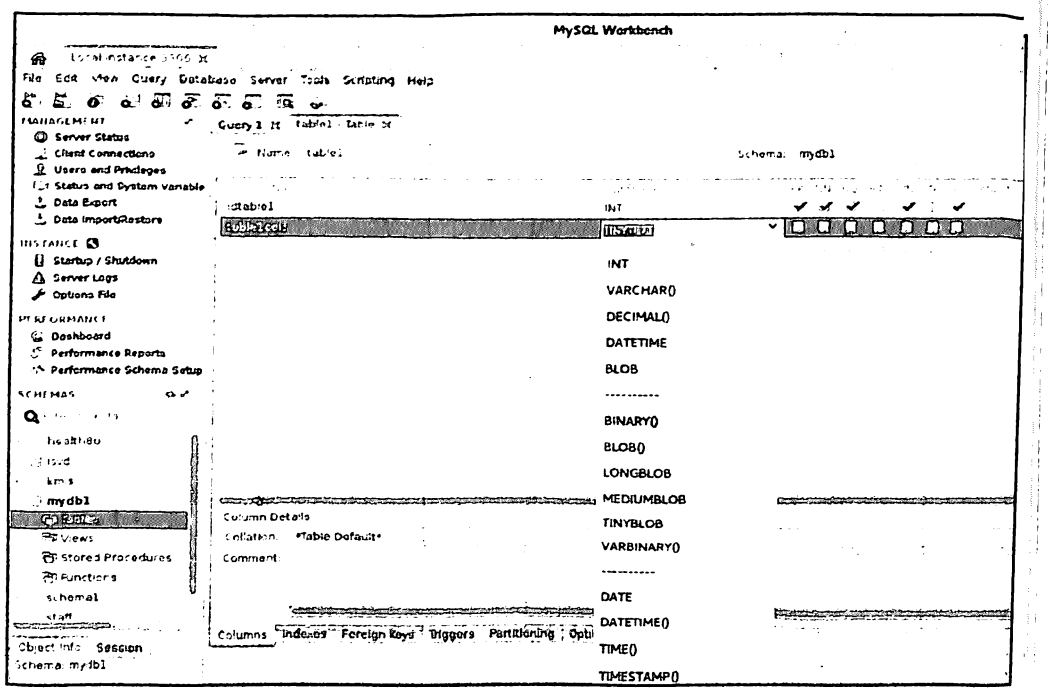


Рис. 1.34. Окно создания таблицы

Напоминаем, что во всех случаях изменений таблиц или базы данных требуется подтвердить свои действия (**Apply**) или отказаться от изменений (**Cancel**).

Создадим новую таблицу с именем **table1** (рис. 1.34).

Далее необходимо создать поля, индексы, внешние ключи, триггеры и т. п.

Внимание! Таблица может быть создана и без ключевого поля, но, к сожалению, ввести данные в нее не получится, т. е. работать с ней будет невозможно.

Обратите внимание на тип данных, который выбирается для каждого поля (см. рис. 1.28). В MariaDB столбцу может быть назначен следующий тип данных:

- **PK: Primary key** — первичный ключ;
- **NN: Not null** — неопределенное значение недопустимо;
- **UQ: Unique** — уникальный;
- **BIN: Binary** — бинарный;
- **UN: Unsigned** — беззнаковый;
- **ZF: Zero fill** — заполнение нулями;
- **AI: Autoincrement** — счетчик (автоинкремент).

Для поля первичного ключа выберем тип данных Integer (подробнее о типе данных MariaDB см. соответствующую документацию), PK, NN, UQ, AI (рис. 1.35).

Создадим поле для ввода символьной (строковой) информации. Назовем его **table1col1** (см. рис. 1.35). Тип данных выберем в раскрывающемся списке, например, TINYTEXT.

Чтобы выйти из режима создания таблицы, можно нажать на крестик (при этом закрывается соответствующая вкладка), после чего появится сообщение с вопросом, сохранить изменения или нет. Если принять изменения (нажата кнопка **Apply**), то появится сообщение (см. рис. 1.35). Обратите внимание, что на выполнение отправляется SQL-скрипт. После выполнения скрипта таблица создана (рис. 1.36).

Внесем данные в созданную таблицу. Для этого надо вызвать режим редактирования таблиц, например, нажав на значок рядом с таблицей (см. рис. 1.13). Обратите внимание, что в окне объектов отображается информация о данной таблице. В строковые поля поместим текст: Строка 1 и Строка 2 соответственно. Поле индекса заполнять не надо, поскольку у нас выбран тип данных автоинкремент (рис. 1.37).

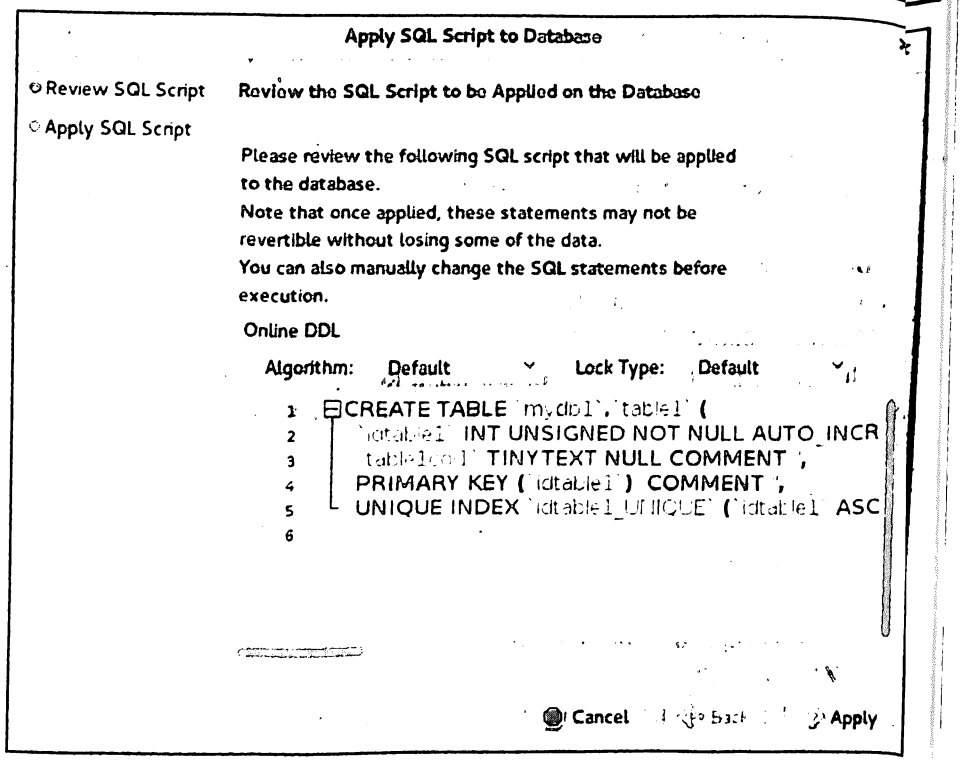


Рис. 1.35. Скрипт для создания таблицы

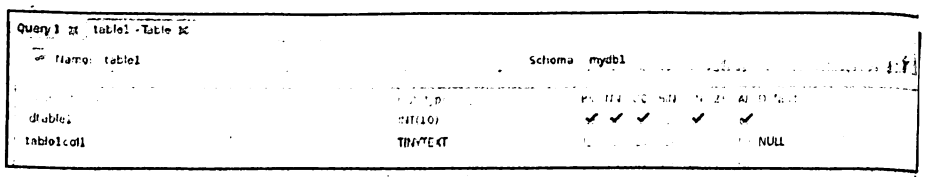


Рис. 1.36. Таблица создана

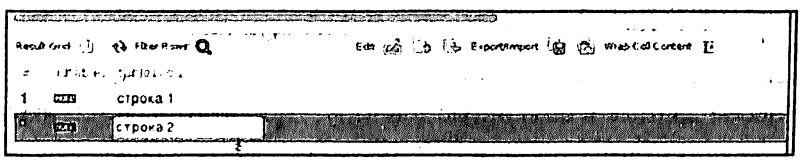


Рис. 1.37. Заполнение таблицы

После этого при внесении данных появится окно сообщения (рис. 1.38) и данные можно принять или отказаться. Обратите внимание, что при выполнении любых запросов генерируется SQL-код.

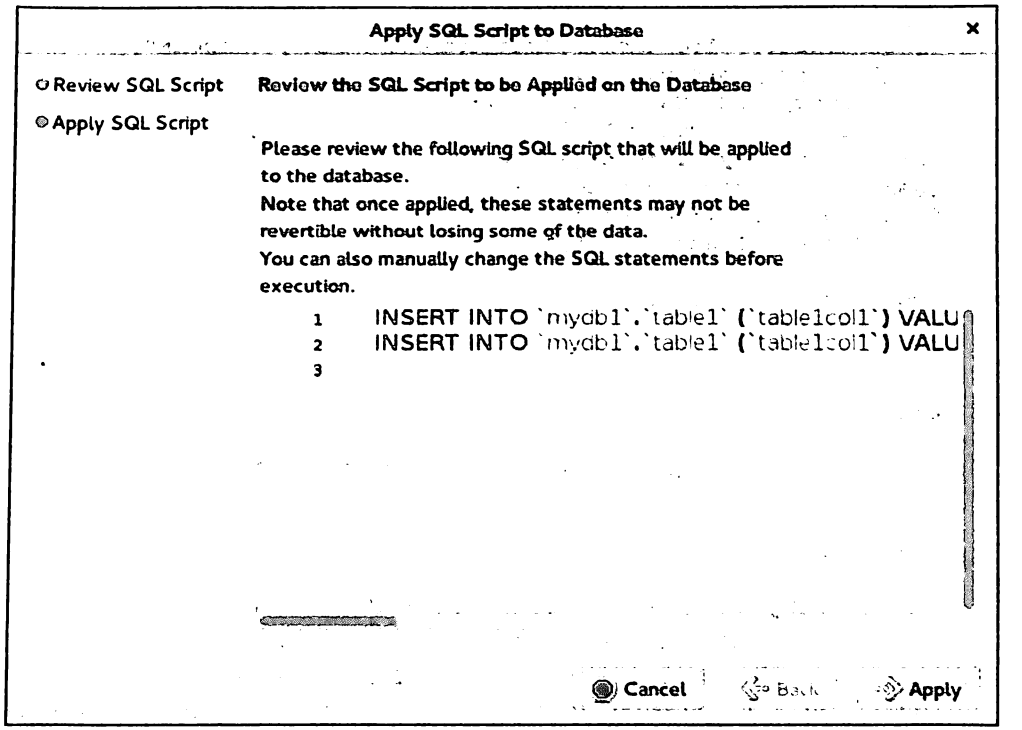


Рис. 1.38. SQL-код внесения данных в таблицу

Окончательный вид таблицы представлен на рис. 1.39.

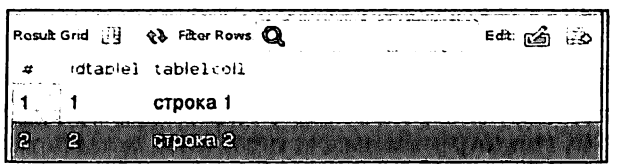


Рис. 1.39. Окончательный вид таблицы

Итак, тестовая база данных из одной таблицы создана. Проверим работоспособность созданной базы данных. Запустим следующий код (HTML5):

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
```

```
<?php
mysql_connect("localhost","имя_пользователя", "пароль");
mysql_select_db("mydb1");
mysql_query("SET NAMES 'utf8'");
$result = mysql_query("select * from table1;");
while($row = mysql_fetch_array($result))
{$sAux1 = "{$row['idtable1']}";
$sAux2 = "{$row['table1col1']}";
echo "$sAux1 $sAux2 <BR>";
}
mysql_free_result($result);
mysql_close();
?>
</body>
</html>
```

На экран будет выведено содержимое таблиц базы данных:

```
1 строка1
2 строка2
```

На панели управления вкладки таблицы имеются пиктограммы для работы с таблицами (рис. 1.40). Описание элементов управления, вызываемых этими пиктограммами, представлено в табл. 1.3: пиктограмма обновления данных, поле фильтра для выборки строк, содержащих последовательности символов, введенных в поле, далее три пиктограммы для редактирования, две пиктограммы для работы с файлами и последняя для изменения размера ячеек (например, в случае длинной строки можно отображать ее полностью или частично).

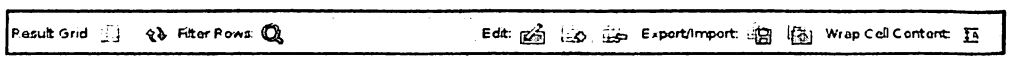
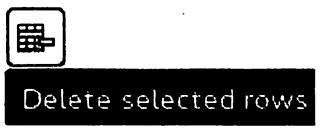
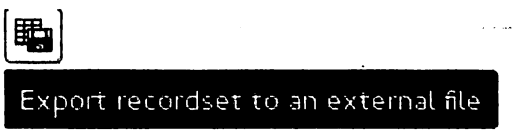
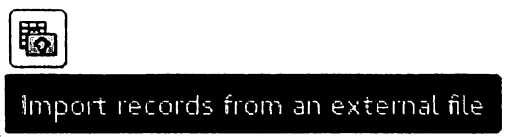


Рис. 1.40. Пиктограммы панели управления вкладки таблицы

Таблица 1.3. **Элементы управления вкладки таблицы**

Элемент управления	Описание
	Устанавливает сортировку колонок по умолчанию
	Служит для обновления данных во вкладке

Окончание табл. 1.3

Элемент управления	Описание
	Редактирование текущей строки
	Вставка новой строки в таблицу
	Удаление выбранной строки из таблицы
	Экспорт записей во внешний файл
	Импорт записей из внешнего файла
	Сворачивание длинного текста в ячейке

Пример работы поля фильтрации приведен на рис. 1.41.

Для отфильтровывания записей необходимо в поле фильтрации ввести значение и нажать Enter. На данном рисунке видно, что в поле введены символы «а 2», следовательно, должны быть отсортированы

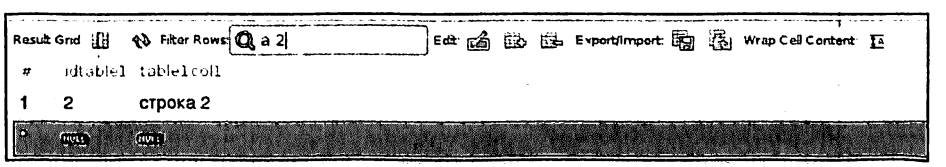


Рис. 1.41. Фильтрация таблицы

строки, в которых содержится такое сочетание символов, т. е. вторая строка, содержащая в текстовом поле значение «а 2».

При нажатии на кнопку редактирования строки, появляется возможность изменять или вносить информацию в редактируемые поля (рис. 1.42).

Для работы со строками также имеется контекстное меню, которое вызывается правой кнопкой мыши (рис. 1.43) и позволяет копировать и удалять строки, устанавливать неопределенное значение NULL и пр.

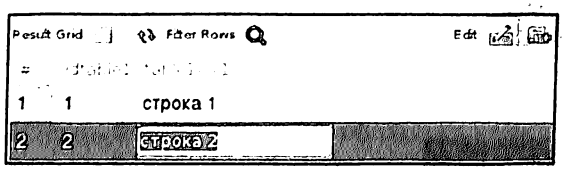


Рис. 1.42. Редактирование таблицы

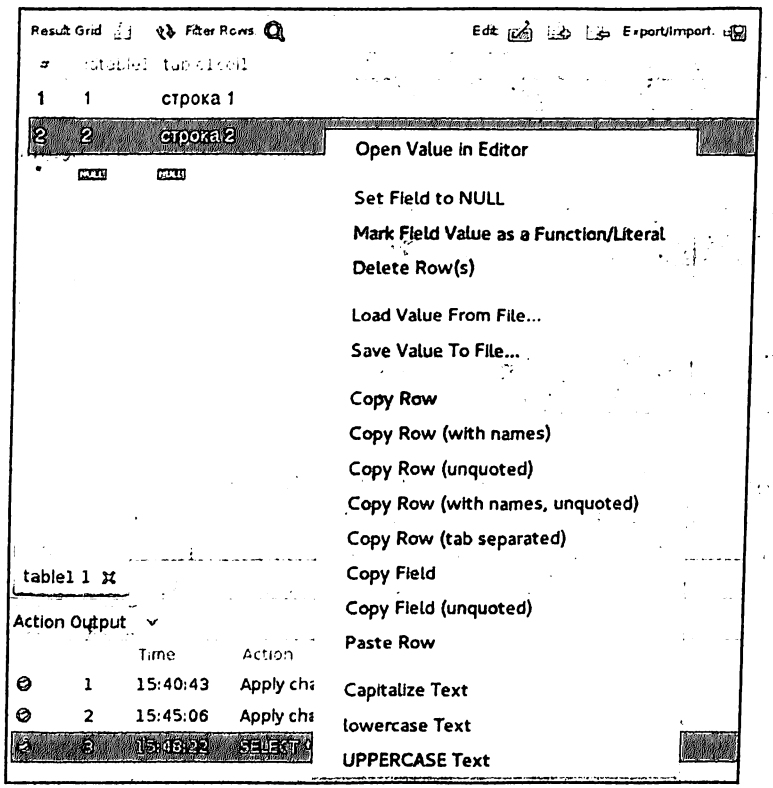


Рис. 1.43. Контекстное меню для работы со строками

В правой части окна находятся кнопки, позволяющие отображать данные в виде таблицы (рис. 1.44), для редактирования (рис. 1.45), в виде полей (рис. 1.46) и т. п.

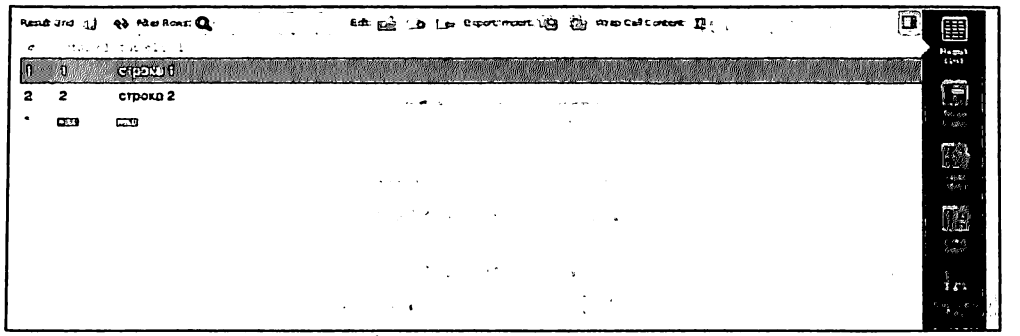


Рис. 1.44. Отображение данных в виде таблицы

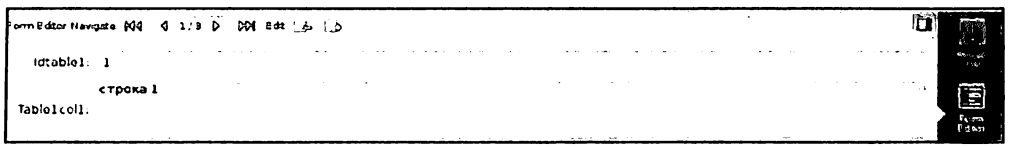


Рис. 1.45. Отображение данных для редактирования

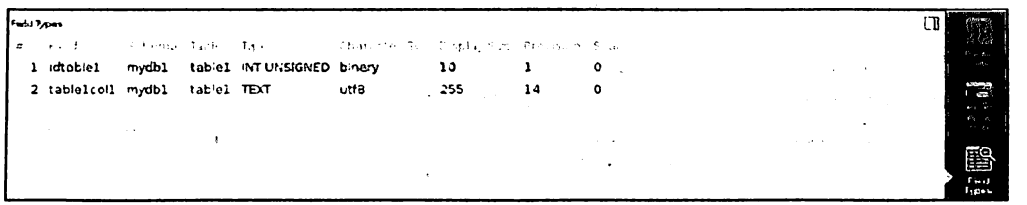


Рис. 1.46. Отображение данных в виде полей

В самом низу окна находятся вкладки **Action Output**, **Text Output** и **History**. Как следует из названия, при переходе на вкладку **Action Output** (рис. 1.47) отображаются действия, совершенные с таблицей. Данная вкладка обеспечивает просмотр сообщений между скриптом

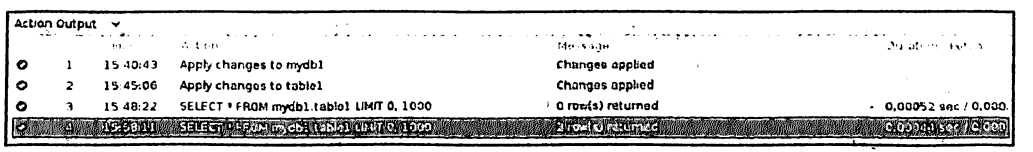


Рис. 1.47. Вкладка **Action Output**

и сервером, в частности, это могут быть сообщения об ошибке. Каждое сообщение показывает время, выполненное действие и ответ сервера. Данная информация особенно полезна для скриптов, содержащих ошибки.

Вкладка **Text Output** (рис. 1.48) представляет текстовый результат запроса.

Рис. 1.48. Вкладка **Text Output**

Вкладка **History** (рис. 1.49) показывает историю выполненных SQL-операций. Время и SQL-код каждого запроса отображается во вкладке. При щелчке мыши на соответствующей дате будет показана история работы с базой за это число.

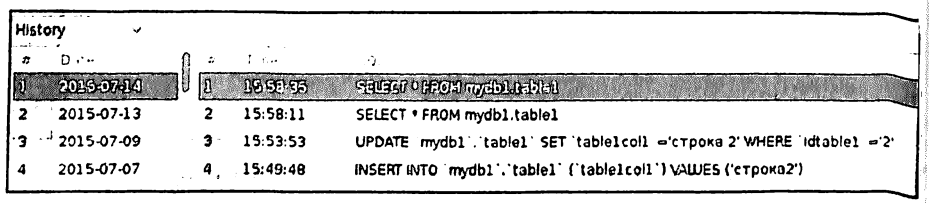


Рис. 1.49. Вкладка **History**

Учащимся предлагается выполнить практическую работу, основанную на изученном теоретическом материале. Выполнение практической работы предполагает изучение рабочего пространства разработчика и создание тестовой базы данных.

Порядок выполнения практической работы

1. Ознакомиться с описанием.
2. Открыть рабочую область программы **MySQL Workbench** для создания базы данных.
3. Создать тестовую базу данных, состоящую из одной таблицы и нескольких столбцов, выбрать для столбцов различные типы данных.
4. Проверить создаваемый программой SQL-скрипт на соответствие структуре созданной базе данных.
5. Сохранить базу данных и выйти из программы.
6. Подготовить отчет по практической работе, содержащий основные этапы работы с тестовой базой данных.

Контрольные вопросы

1. Какие способы создания баз данных имеются в **MySQL Workbench**?
2. Какие пункты меню имеются в рабочей области? Для чего они предназначены?
3. Для каких объектов можно вызвать контекстное меню?
4. Для чего служит вкладка **Output Action**? **History**?
5. Перечислите основные шаги для создания таблицы в **MySQL Workbench**.
6. Какие элементы управления предназначены для работы с таблицей во вкладке **table**?

Лабораторная работа 2
ПРАКТИЧЕСКАЯ РАБОТА С MYSQL WORKBENCH SQL EDITOR ДЛЯ ПОСТРОЕНИЯ РЕАЛЬНОЙ БАЗЫ ДАННЫХ И ЗАПРОСОВ К НЕЙ

Цель: формирование навыков по анализу предметной области, построению инфологической модели и работе с инструментом MySQL Workbench для создания баз данных при помощи SQL Editor.

Анализ предметной области и построение инфологической модели

Первым этапом проектирования баз данных является анализ предметной области. Выберем для примера предметную область «запись студентов на курсы дисциплин по выбору». Создадим для нее небольшую базу данных (предполагается, что читатель уже знаком с основами теории реляционных баз данных). Для простоты будем считать, что база данных находится, по крайней мере, в третьей нормальной форме, а в качестве первичного ключа предлагается использовать либо счетчик, либо целое число.

Поставим задачу следующим образом. Предположим, что на факультете, помимо основных дисциплин, студенты до окончания учебы должны набрать определенное число часов дисциплин по выбору.

Исходя из анализа предметной области можно выделить следующие сущности (таблицы баз данных): **student** (Студент), **electives** (Дисциплина), которые связаны между собой отношением «многие-ко-многим», поскольку каждый студент может записаться на различные курсы дисциплин, а каждый курс дисциплин может посещаться различными студентами. Однако в реляционной модели данных следует заменить отношение «многие-ко-многим» на несколько отношений «один-ко-многим». Для этого требуется добавить еще одну сущность (таблицу), которая была бы связана с обеими сущностями, но с помощью отношения «один-ко-многим». Таковой сущностью (таблицей) может быть процесс «фиксации» записи студентов на курс дисциплины по выбору — **registration** (Запись).

Далее следует определить атрибуты каждой сущности. К сущности Студент относятся такие атрибуты, как имя, кафедра, год поступления, телефон. К сущности Дисциплина по выбору — название и количество часов. К сущности Запись — дата (год записи).

Создадим еще одну таблицу **cafedra** (Кафедра), которая связана с таблицей **student** отношением один-ко-многим. Для простоты базы данных не будем вводить таблицы преподавателей и устанавливать связи между преподавателями, кафедрами и дисциплинами.

Выделим атрибуты, которые однозначно определяют каждый экземпляр сущности, — первичные ключи. Поскольку ни в одной из описанных выше сущностей ни один из атрибутов, выявленных ранее, не подходит в качестве первичного ключа, введем для каждой таблицы первичные ключи, которые будут счетчиками, а по типу данных — целыми числами.

Внимание! Как было сказано выше, недопустимо создавать таблицу без первичного ключа, поскольку работа с ней невозможна.

Создание таблицы

После выделения сущностей и атрибутов предметной области можно перейти к созданию таблиц базы данных. Как известно, основу реляционных баз данных составляют двумерные таблицы, предназначенные для хранения информации. Таблица в реляционной базе данных состоит из строк (записей) и столбцов — полей с определенным типом данных. Ранее было показано, каким образом для столбца задается тип данных. Поскольку для идентификации каждого экземпляра сущности необходим первичный ключ, каждая таблица базы данных должна содержать хотя бы один столбец. Однако поскольку практический смысл в такой таблице отсутствует, то каждая таблица базы данных должна содержать хотя бы два столбца, один из которых — первичный ключ, второй — информационное поле.

При создании таблицы необходимо определить:

- название таблицы;
- названия столбцов таблицы;

- тип данных каждого столбца;
- столбцы, входящие в состав первичного ключа;
- столбцы (поля) таблицы, обязательные для заполнения;
- размер памяти для хранения каждого столбца.

Базовый синтаксис оператора создания таблицы имеет следующий вид:

```
CREATE TABLE <table name>
( { <column name> <data type> | <size>]
[<colcnstrnt> ...]} .,.. );
[<tabconstrnt>] .,.. );
```

Элементы, используемые в команде CREATE TABLE, приведены в табл. 2.1.

Таблица 2.1. **Элементы, используемые в команде CREATE TABLE**

Элемент	Определение
<table name>	Имя таблицы, создаваемой этой командой
<column name>	Имя столбца таблицы
<data type>	Тип данных, которые могут содержаться в столбце
<size>	Размер. Его значение зависит от **<data type>**
<colconstrnt>	Может быть любым из следующих: **NOT NULL** (НЕ НУЛЕВОЙ), **UNIQUE** (УНИКАЛЬНЫЙ), **PRIMARY KEY** (ПЕРВИЧНЫЙ КЛЮЧ), **CHECK(<predicate>)** (ПРОВЕРКА предиката), **DEFAULT = <value expression>** (ПО УМОЛЧАНИЮ = значимому выражению), **REFERENCES <table name> [(<column name> .,..)]** (ССЫЛКА НА имя таблицы [(имя столбца)])
<tabconstrnt>	Может быть любым из следующих: **UNIQUE** (УНИКАЛЬНЫЙ), **PRIMARY KEY** (ПЕРВИЧНЫЙ КЛЮЧ), **CHECK** (ПРОВЕРКА предиката), **FOREIGN KEY** (ВНЕШНИЙ КЛЮЧ), **REFERENCES <table name> [(<column name> .,..)]** (ССЫЛКА НА имя таблицы [(имя столбца)])

Ключевое слово **NULL** используется для указания того, что в данном столбце могут содержаться значения NULL. Значение NULL отличается от пробела или нуля и указывает, что данные недоступны, опущены или недопустимы. Если указано ключевое слово NOT NULL, то будут отклонены любые попытки поместить значение **NULL** в данный столбец. Если указан параметр **NULL**, помещение значений **NULL** в столбец разрешено. По умолчанию стандарт SQL предполагает наличие ключевого слова **NULL**.

Изменение таблицы

Структура существующей таблицы может быть модифицирована с помощью команды ALTER TABLE, синтаксис которой представлен ниже:

```
ALTER TABLE <table name> ADD <column name> <data type> <size>;
```

Столбец будет добавлен со значением **NULL** для всех строк таблицы, поскольку добавляемый столбец не может быть определен с атрибутом NOT NULL. Этот атрибут означает, что для каждой строки данных соответствующий столбец должен содержать некоторое значение, поэтому добавление столбца с атрибутом NOT NULL приводит к появлению вопроса, как быть с уже существующими строками таблицы, которые в данном столбце значений не имеют.

Для добавления обязательных полей в существующую таблицу следует:

- добавить в таблицу новый столбец, определив его с атрибутом **NULL**;
- ввести в новый столбец какие-либо значения для каждой строки данных таблицы;
- изменить структуру таблицы, заменив атрибут этого столбца на **NOT NULL**.

Новый столбец станет последним по порядку столбцом таблицы. Можно добавить сразу несколько новых столбцов в одной команде, отделив их запятыми. Имеется также возможность удалять или изменять столбцы. Тем не менее необходимо убедиться, что добавляемые поля не вступают в противоречие с уже созданными таблицами базы данных, а пользователь, который использует данную команду, имеет необходимые права доступа. Именно поэтому использовать **ALTER TABLE** следует только в том случае, когда это не нарушает логику и целостность данных.

Удаление таблицы

При разработке баз данных возникают ситуации, когда необходимо удалить таблицу. Это можно сделать при помощи следующего оператора:

```
DROP TABLE <table name>имя_таблицы [RESTRICT | CASCADE]
```

Следует отметить, что эта команда удалит не только указанную таблицу, но и все входящие в нее записи (данные). Если необходимо удалить данные из таблицы, сохранив при этом ее структуру, то следует воспользоваться командой **DELETE**.

```
DELETE FROM <table name>
[ WHERE <predicate>
| WHERE CURRENT OF <cursor name> (*только для вложения*) ];
```

Оператор **DROP TABLE** дополнительно позволяет указывать, как именно выполняется операция удаления. Если в операторе указано ключевое слово **RESTRICT**, то при наличии в базе данных хотя бы одного объекта, существование которого зависит от удаляемой таблицы, выполнение оператора **DROP TABLE** будет отменено. Если указано ключевое слово **CASCADE**, автоматически будут удалены все объекты базы данных, чье существование зависит от удаляемой таблицы, а также другие объекты, зависящие от удаляемых объектов. Таким образом, удаление может затронуть значительное количество данных, поэтому данный оператор следует использовать с максимальной осторожностью и следить за установкой прав доступа пользователей, работающих с базой данных, чтобы ограничить круг лиц, имеющих право вносить изменения.

Чаще всего оператор **DROP TABLE** используется для исправления ошибок, допущенных при создании таблицы. Если таблица была создана с некорректной структурой, можно воспользоваться оператором **DROP TABLE** для ее удаления, после чего создать таблицу заново.

Создание базы данных для описанной предметной области

Выполнение работы следует, как и в предыдущем случае, начать с входа в оболочку **MySQL Workbench**. Вызов программы осуществляется щелчком по соответствующей иконке на рабочем столе или через меню. Далее следует выбрать **Open Connection to Start Querying** (рис. 2.1).

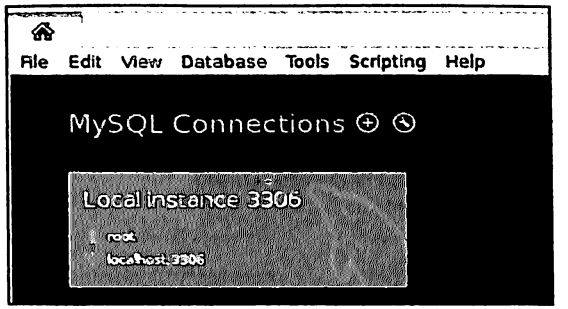


Рис. 2.1. Вход в **MySQL Workbench** для создания базы данных

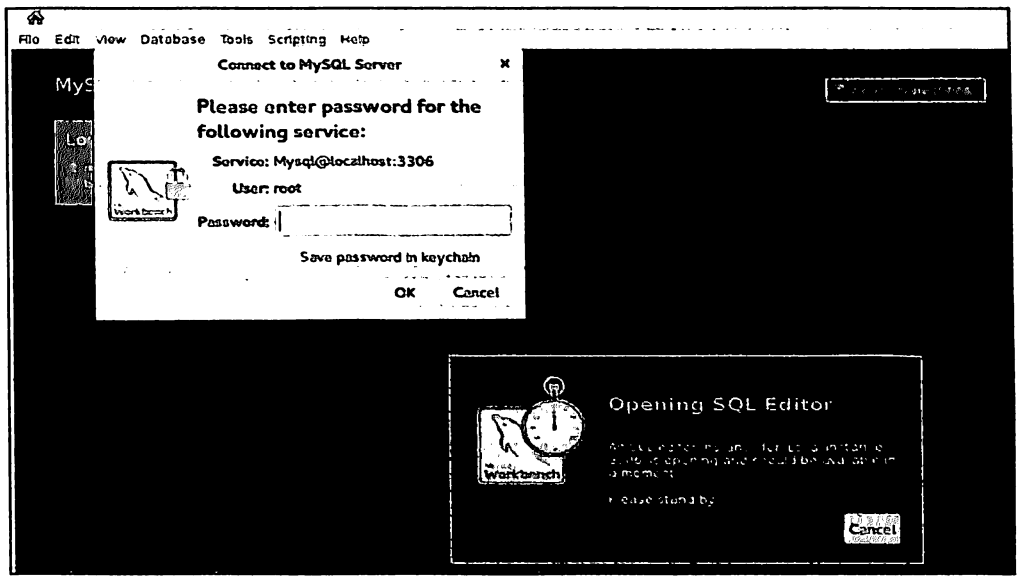


Рис. 2.2. Ввод пароля

Далее, если это необходимо, следует ввести пароль (рис. 2.2). Обратите внимание, что в данном случае работа производится в роли администратора, т. е. из-под **root**. Такой режим при реальной эксплуатации СУБД не является предпочтительным, в том числе с точки зрения безопасности, поэтому желательно в дальнейшем создавать пользователей с определенными ролями. Как это сделать, будет сказано ниже. На данном этапе для простоты будем работать из-под **root**.

Затем производятся настройки свойств. Сделать это можно при помощи выбора пункта меню **Edit → Preferences**. На общей вкладке

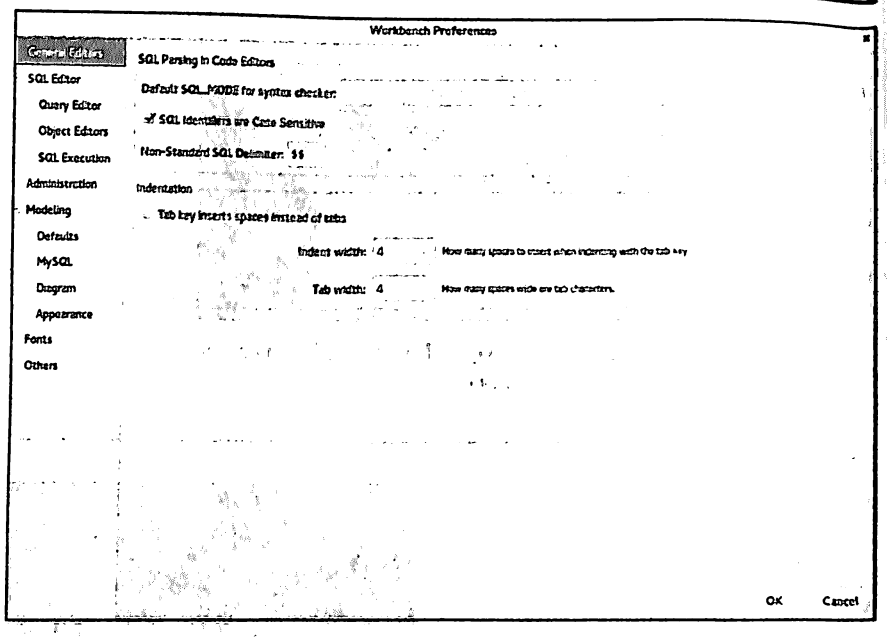


Рис. 2.3. Вкладка **General** меню **Preferences**

(рис. 2.3) настраиваются общие свойства: интервал автосохранения, свойства истории и т. п.

Следует помнить, что выбор большого значения для сохраняемой истории значительно увеличивает объем памяти и приводит к замедлению работы системы.

Вкладка **SQL Editor** (рис. 2.4) предназначена для редактирования свойств редактора. Она имеет несколько дополнительных вкладок **Query Editor**, **Object Editor** и **SQL Execution**. При помощи выбора опций в этих вкладках можно произвести настройки, будут ли различаться идентификаторы, имеющие одинаковое написание, но с буквами в разном регистре, настройки свойств запросов: максимальной длины истории, отображения максимальной длины поля в байтах и т. п. (см., например, рис. 2.5, вкладка **SQL Execution**).

Рядом с каждым пунктом меню находятся пояснения, какие именно значения выбираются в данном пункте (см. рис. 2.4).

Вкладка **Administrator** предназначена для администрирования. В том случае, если пользователь не планирует заниматься администрированием, следует оставить в ней настройки, заданные по умолчанию.

Вкладка **Model** предназначена для настройки свойств моделей. В частности, в ней по умолчанию настроены параметры первичного

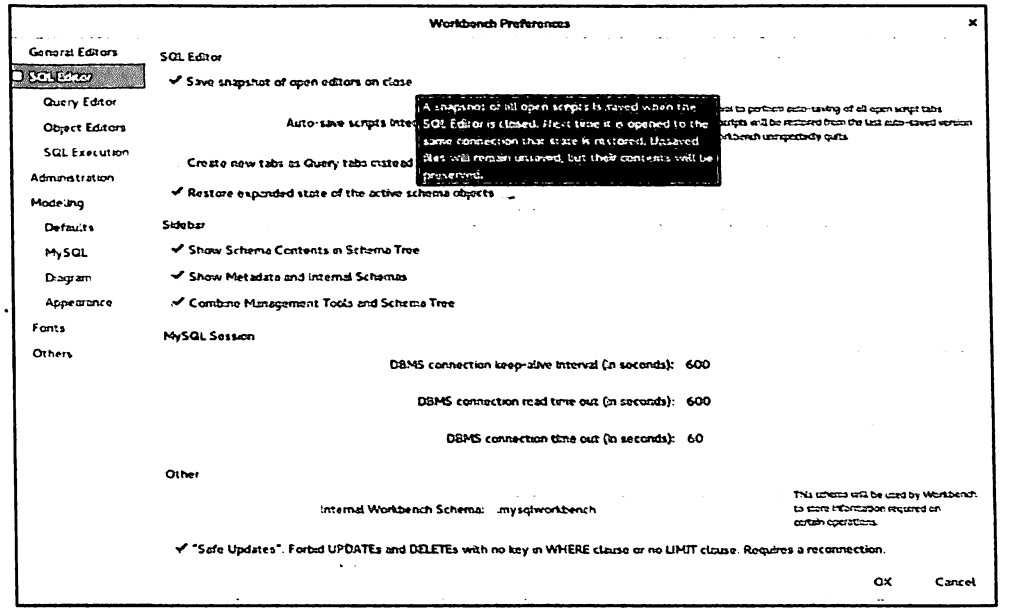


Рис. 2.4. Вкладка **SQL Editor**

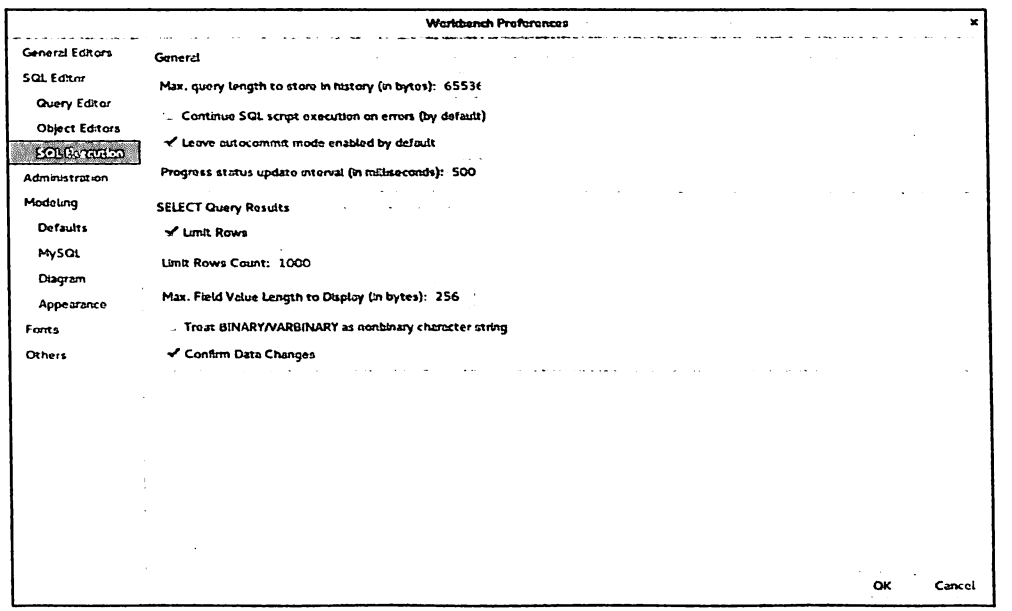


Рис. 2.5. Вкладка **SQL Execution**

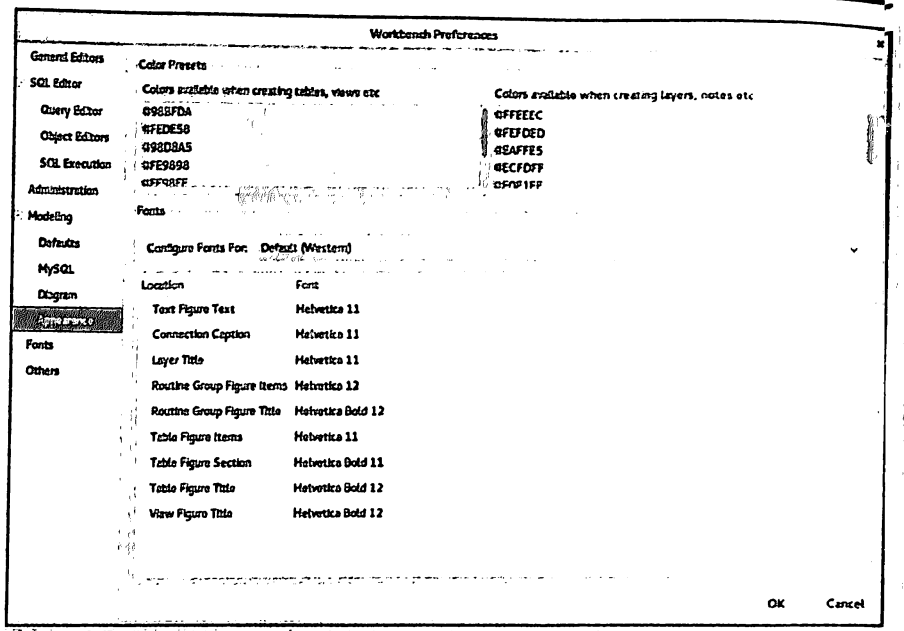


Рис. 2.6. Вкладка Appearance

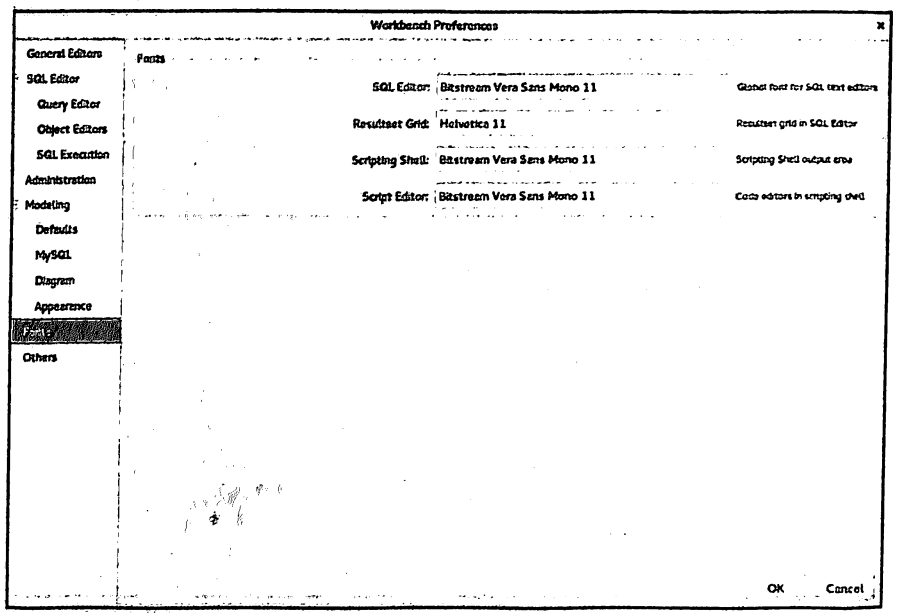


Рис. 2.7. Вкладка Fonts

ключа (**PK**) и есть возможности по настройке ссылочной целостности при изменении и удалении данных: **ON UPDATE** и **ON DELETE** (см. приложение А). Можно также настроить свойства диаграмм. Для настройки шрифтов в моделях, в том числе для диаграмм, служит вкладка **Appearance** в разделе **Model** (рис. 2.6).

Для настройки шрифтов служит вкладка **Fonts** (рис. 2.7).

Вкладка **Others** позволяет произвести настройки доступа по защищенному протоколу повторного использования соединения HTTP и установить его тайм-аут.

После настройки свойств перейдем непосредственно к созданию базы данных.

Создадим новую базу данных с названием **student**. Произведем ее настройки, как было представлено ранее (рис. 2.8).

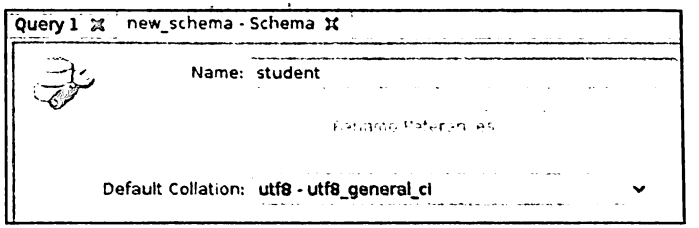


Рис. 2.8. Создание базы данных **student**

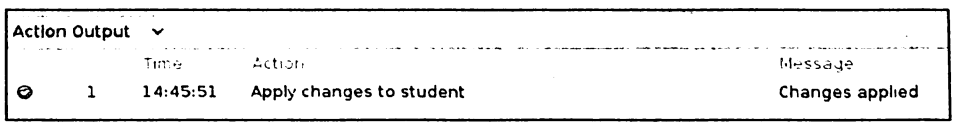


Рис. 2.9. База данных создана (изменения приняты)

Как было показано ранее в главе 1, после нажатия кнопки **Apply** появится окно с SQL-кодом. Нажав еще раз **Apply**, закончим операцию. При успешном выполнении операции во вкладке **Action Output** появится сообщение об успешном выполнении (рис. 2.9) и название базы данных появится слева в окне объектов (рис. 2.10).

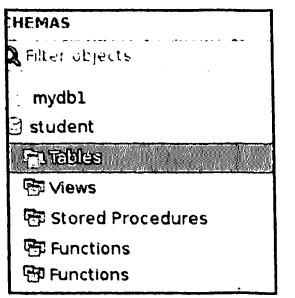


Рис. 2.10. Название новой базы данных в окне объектов

Создадим таблицу **student** (студент). Для этого вспомним порядок действий, который был осуществлен при выполнении тестового

примера. Например, щелкнув на слове «**Tables**» правой кнопкой мыши, выберем в контекстном меню пункт «Create Table...» и создадим таблицу.

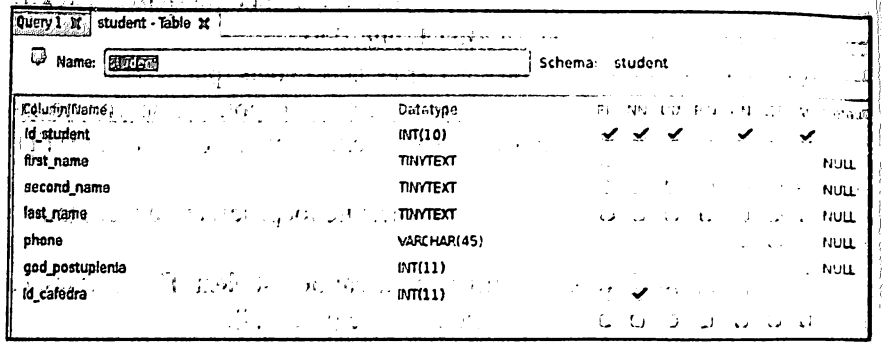


Рис. 2.11. Таблица **student (студент)**

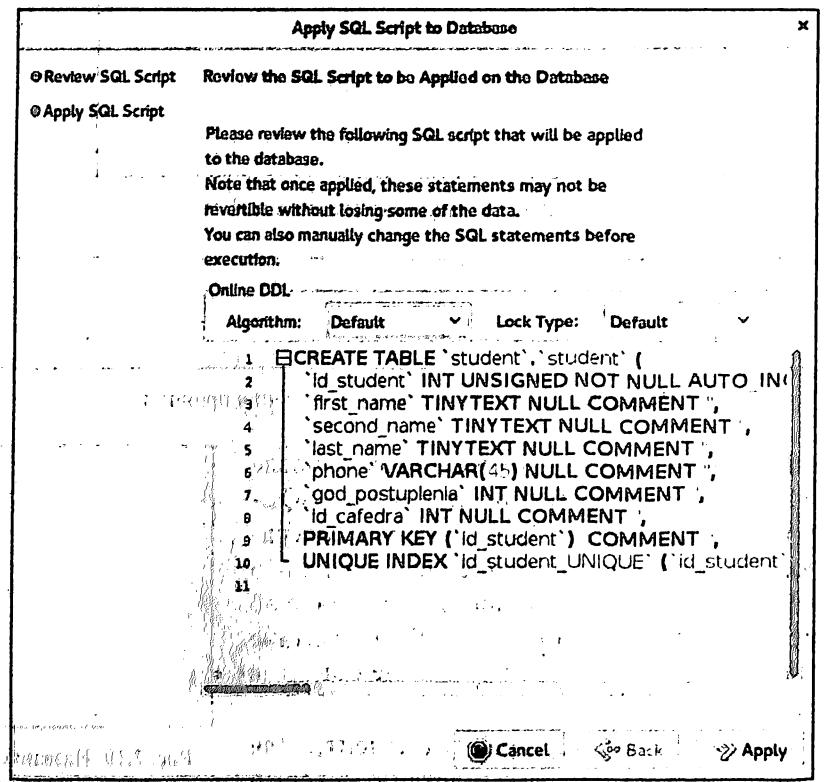


Рис. 2.12. SQL-код создания таблицы **student (студент)**

Заметим, что поскольку название полей являются именами переменных, зададим название таблиц и полей латиницей.

Для простоты создадим суррогатный первичный ключ, чтобы гарантировать третью нормальную форму базы данных (рис. 2.11). Сделаем его счетчиком (автоинкрементом). Для первичного ключа будет использоваться тип данных — целое (**INT**).

Далее необходимо определить помимо первичного ключа тип данных для остальных атрибутов. Очевидно, что остальные поля являются текстовыми. Определим для них тип данных **TINYTEXT**. Процедура выбора типа данных также полностью аналогична процедуре создания тестового примера. Для завершения процесса создания атрибутов вспомним, что студент учится на некоторой кафедре, и добавим в таблицу внешний ключ **id_cafedra**. Далее необходимо нажать кнопку **Apply** внизу страницы для фиксации изменений. В результате получим таблицу (см. рис. 2.11).

Как и в случае тестового примера, будет сгенерирован SQL-код (рис. 2.12).

Возможны различные способы создания таблиц, однако написание SQL-скриптов рекомендуется только для продвинутых пользователей, поскольку работа с графическим редактором значительно нагляднее.

Аналогичным образом создаются таблицы **cafedra** (кафедра) (рис. 2.13), **electives** (дисциплины по выбору) (рис. 2.14) и **registration** (регистрация) (рис. 2.15).

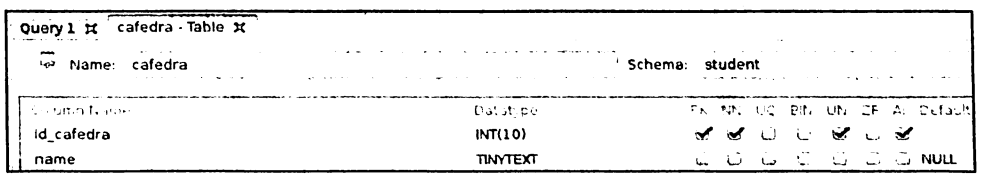


Рис. 2.13. Таблица **cafedra (кафедра)**

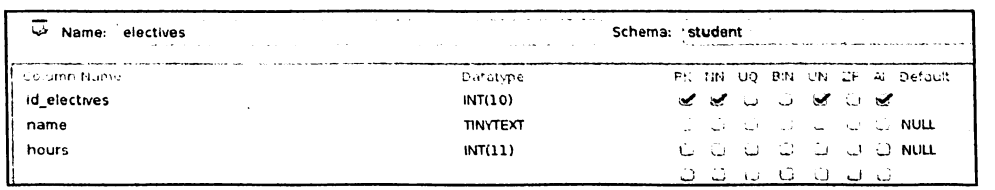


Рис. 2.14. Таблица **electives (дисциплины по выбору)**

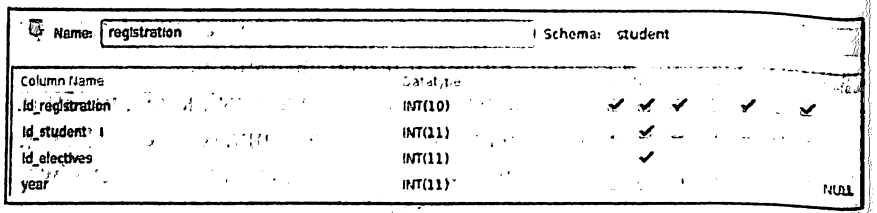


Рис. 2.15. Таблица **registration (регистрация)**

Обратите внимание на типы данных. Для полей, где возможны текстовые данные до 255 символов, использовался тип данных **TINYTEXT**. Для номера телефона — **VARCHAR**. Для простоты расчетов год будет иметь тип не Дата/Время, а целое число **INT**.

Создание базы данных завершено.

Следующим шагом является внесение данных в таблицу.

Сначала внесем данные в таблицы **кафедра** и **дисциплины по выбору** (рис. 2.16 и 2.17), например, нажав на значок: рядом с таблицей (подробнее об этом сказано выше). Поскольку ключевое поле имеет тип данных автоинкремент (счетчик), оно заполняется не пользовате-

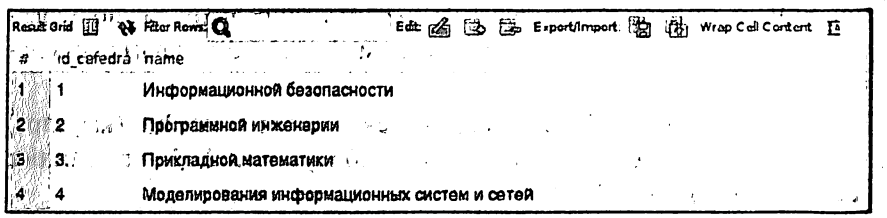


Рис. 2.16. Внесение данных в таблицу **cafedra (кафедра)**

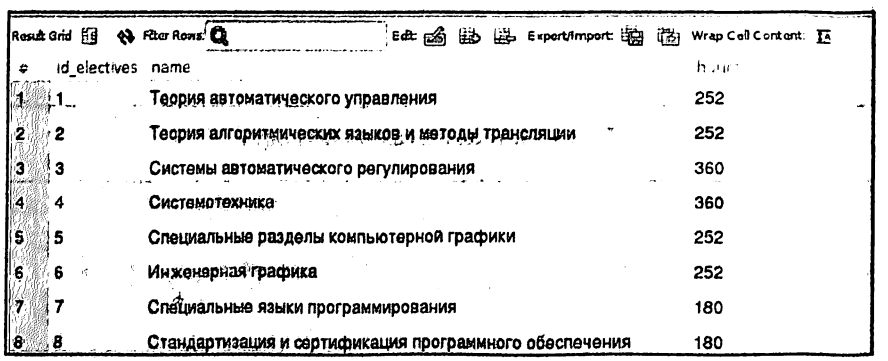


Рис. 2.17. Внесение данных в таблицу **electives (дисциплины по выбору)**

лем, а автоматически. При нажатии на кнопку **Apply** генерируется SQL-скрипт.

Если необходимо произвести настройки, то внизу окна существуют вкладки для работы со столбцами (Columns), индексами (Idexes), внешними ключами (Foreing Keys), триггерами (Triggers), для разделения (в том числе по ключу) и с другими опциями (Options). Например, если необходимо изменить значение автоинкремента, то это можно сделать в соответствующем поле вкладки Options (рис. 2.18).

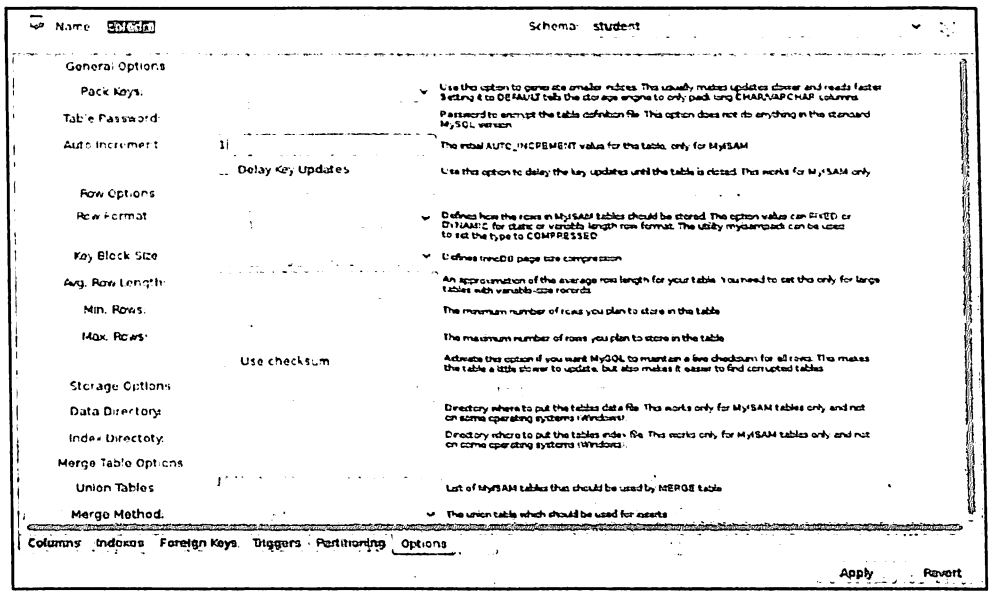


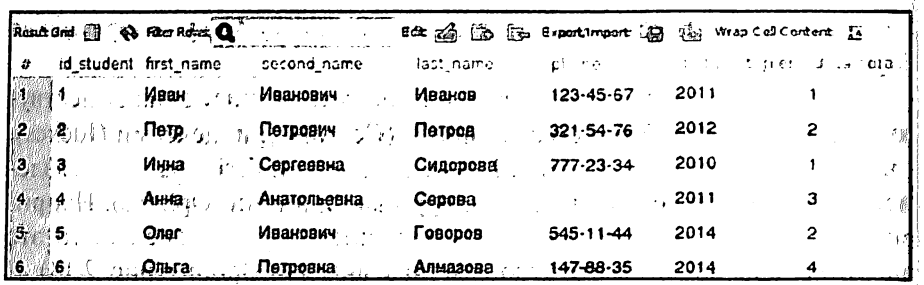
Рис. 2.18. Вкладка **Options**

Теперь внесем данные в таблицу студент. В нашем случае база данных называется **student**, так же как и таблица студент (**student**), что не мешает СУБД функционировать корректно.

Окончательный вид таблицы **student** с внесенными данными представлен на рис. 2.19.

Если необходимо отредактировать уже имеющиеся данные, то порядок действий аналогичен.

После того как данные о студентах, кафедрах и дисциплинах по выбору внесены, можно внести данные в таблицу регистрации (рис. 2.20).



#	id_student	first_name	second_name	last_name	phone	ged_postupleni	id_cafedra
1	1	Иван	Иванович	Иванов	123-45-67	2011	1
2	2	Петр	Петрович	Петров	321-54-76	2012	2
3	3	Инна	Сергеевна	Сидорова	777-23-34	2010	1
4	4	Анна	Анатольевна	Серова		2011	3
5	5	Олег	Иванович	Говоров	545-11-44	2014	2
6	6	Ольга	Петровна	Алмазова	147-88-35	2014	4

Рис. 2.19. Заполненная данными таблица **student**

#	id_registration	id_student	id_electives	year
1	1	1	1	2014
2	2	1	2	2014
3	3	1	3	2015
4	4	2	7	2015
5	5	3	6	2014
6	6	3	3	2015
7	7	4	1	2014
8	8	4	6	2015

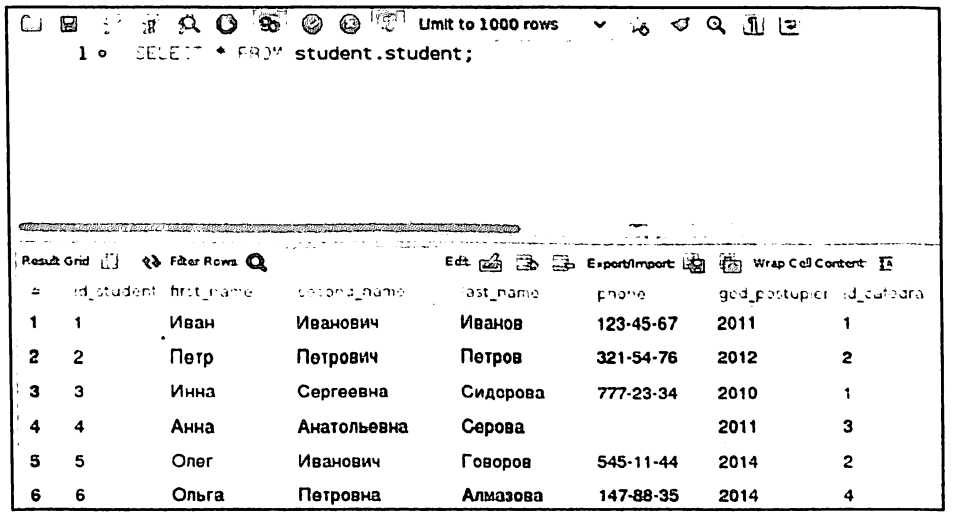
Рис. 2.20. Заполненная данными таблица **registration**

Теперь можно рассмотреть примеры работы запросов к базе данных на основе оператора **SELECT**. Запрос создается в верхней части окна редактора. Отправка запроса на выполнение осуществляется при помощи кнопки ⚡ на панели инструментов.

Для того чтобы составить список сведений обо всех студентах, необходимо выполнить следующий оператор:

```
SELECT * FROM student.student;
```

Будет выведена таблица **student** (студент) (рис. 2.21). Обратите внимание, что поскольку может существовать несколько баз данных, то в некоторых случаях необходимо явно указать, из какой базы данных выбирается таблица. Указанное делается следующим образом: сначала указывается имя базы данных, затем через точку имя таблицы: **student.student**. Данная версия MySQL Workbench позволяет не указывать имя базы данных перед таблицей, если база данных активна.



```
1 • SELECT * FROM student.student;
```

#	id_student	first_name	second_name	last_name	phone	ged_postupleni	id_cafedra
1	1	Иван	Иванович	Иванов	123-45-67	2011	1
2	2	Петр	Петрович	Петров	321-54-76	2012	2
3	3	Инна	Сергеевна	Сидорова	777-23-34	2010	1
4	4	Анна	Анатольевна	Серова		2011	3
5	5	Олег	Иванович	Говоров	545-11-44	2014	2
6	6	Ольга	Петровна	Алмазова	147-88-35	2014	4

Рис. 2.21. Выборка всех данных из таблицы **student**

Также обратите внимание, что можно использовать одинарные кавычки, что особенно актуально при сравнении полей: текстовые поля необходимо заключать в одинарные кавычки, а числовые — нет.

Еще раз отметим, что если на сервере имеется больше одной базы данных, то необходимо явно указывать, из какой базы данных выбирается таблица, поскольку имена таблиц могут совпадать в различных базах.

Использование одинарных кавычек для названия базы данных и таблиц в данном случае не является обязательным, оно необходимо при составлении динамических запросов — таких, при которых запросы работают на основании поступающих в скрипт данных. Чаще всего это делается для сохранения единообразия скриптов. Далее в изложении названия базы данных и таблиц в кавычки заключаться не будут. Однако использование кавычек в случае подстановки в скрипт строковых данных и данных типа «дата» является обязательным. Наличие в конце запроса точки с запятой не является обязательным.

После того как запрос создан, его можно сохранить. Для этого необходимо нажать на значок 💾 на верхней панели окна запросов (второй слева) и в открывшемся окне (рис. 2.22) выбрать директорию, в которой сохранить скрипт в файле с расширением **.sql**.

Если затем необходимо выполнить запрос, следует выбрать пункт меню «Открыть файл», выбрать файл, в котором хранится скрипт за-

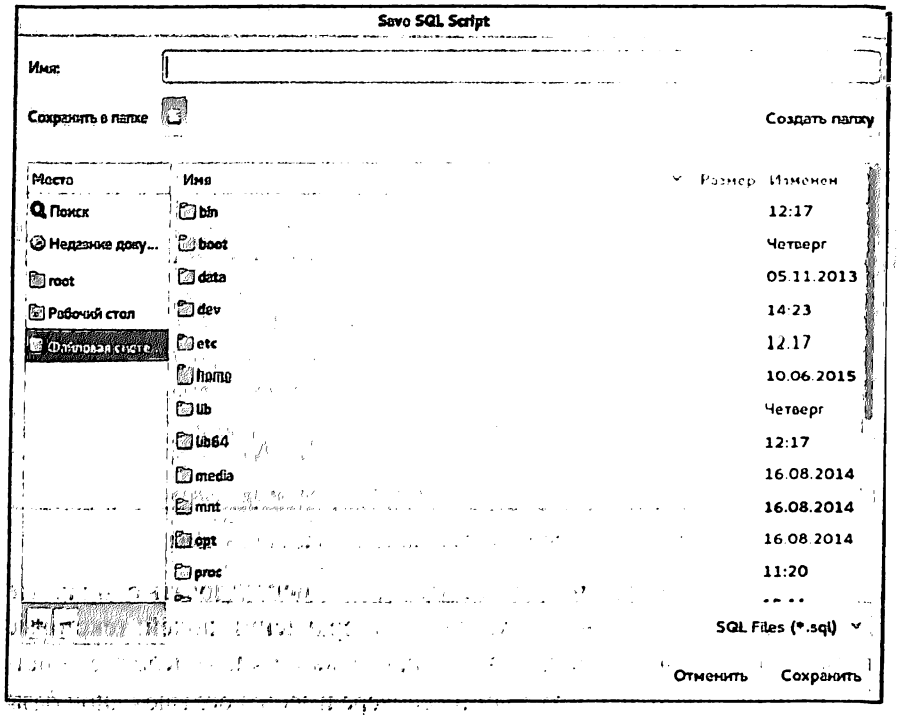


Рис. 2.22. Выбор директории для сохранения скрипта

проса (рис. 2.23), после чего sql код запроса появится в верхней части окна редактора и будет готов к выполнению

С помощью параметра **WHERE** пользователь может указать, какие блоки данных из приведенных в списке **FROM** таблиц появятся в результате запроса. Например, вывод списка студентов кафедры программной инженерии осуществляется при помощи следующего программного кода:

```
SELECT student.last_name, student.first_name,
student.second_name
FROM cafedra INNER JOIN student ON cafedra.id_cafedra =
student.id_cafedra
WHERE cafedra.name="Программной инженерии";
```

За ключевым словом **WHERE** может следовать перечень **условий поиска (предикатов)**, определяющих строки, которые должны быть выбраны при выполнении запроса. Напомним, что предикаты — это выражения, принимающие истинностное значение. Предикаты могут

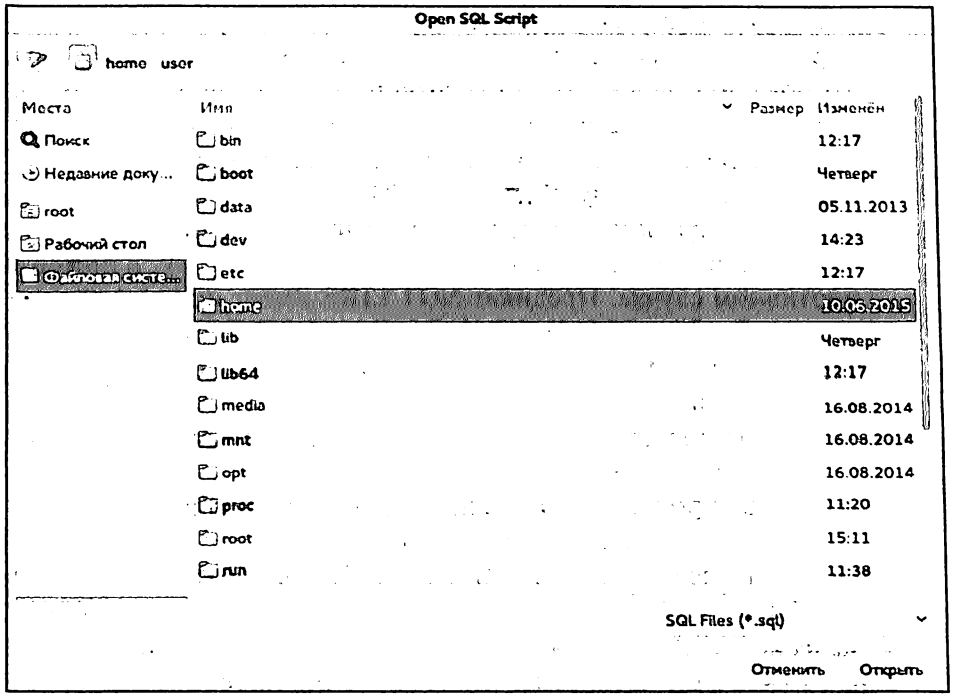


Рис. 2.23. Выбор файла со скриптом для выполнения

быть как одним выражением, так и комбинацией из выражений, строящейся с использованием булевых операторов **AND**, **OR** или **NOT**. Кроме того, здесь может использоваться SQL-оператор **IS**, а также круглые скобки (для определения порядка выполнения операций).

Рассмотрим предикаты:

- **сравнение**. Предикаты сравнения { =, <>, >,<, >=,<= } имеют традиционный смысл, сравниваются результаты вычисления одного выражения с результатами вычисления другого;
- **диапазон**. Предикат **Between A and B** — истинен, когда сравниваемое значение попадает в заданный диапазон, включая границы диапазона. Одновременно в стандарте задан и противоположный предикат **Not Between A and B**, который истинен тогда, когда сравниваемое значение не попадает в заданный интервал, включая его границы. Проверяется, попадает ли результат вычисления выражения в заданный диапазон значений;
- **принадлежность множеству**. Предикат вхождения в множество **IN** (множество) истинен тогда, когда рассматриваемое значение

входит в множество заданных значений. Одновременно в стандарте задан и противоположный предикат **NOT IN** (множество), который истинен тогда, когда сравниваемое значение не входит в заданное множество. Проверяется, принадлежит ли результат вычислений выражения заданному множеству значений;

- **соответствие шаблону**. Предикат **LIKE** требует задания шаблона, с которым сравнивается заданное значение, предикат истинен, если сравниваемое значение соответствует шаблону, и ложен в противном случае. Предикат **NOT LIKE** имеет противоположный смысл. Проверяется, отвечает ли некоторое строковое значение заданному шаблону. По стандарту в шаблон могут быть включены специальные символы:
  - — символ подчеркивания (_) — для обозначения любого одиночного символа;
  - — символ процента (%) — для обозначения любой произвольной последовательности символов;
  - — остальные символы, заданные в шаблоне, обозначают самих себя;
- **значение NULL**. Предикат сравнения с неопределенным значением IS NULL. Неопределенное значение интерпретируется в реляционной модели как значение, неизвестное на данный момент времени. При сравнении неопределенных значений не действуют стандартные правила сравнения: одно неопределенное значение никогда не считается равным другому неопределенному значению. Для выявления равенства значения некоторого атрибута неопределенному применяют специальные стандартные предикаты:

```
<имя атрибута>IS NULL
<имя атрибута> IS NOT NULL
```

Если в данной строке указанный атрибут имеет неопределенное значение, то предикат **IS NULL** принимает значение **TRUE**, а предикат **IS NOT NULL** — **FALSE**, в противном случае предикат **IS NULL** принимает значение **FALSE**, а предикат **IS NOT NULL** принимает значение **TRUE**.

Предикаты существования **EXISTS** и несуществования **NOT EXISTS** относятся к встроенным подзапросам, поэтому в данном изложении не рассматриваются.

Рассмотрим примеры использования предикатов.

Пример использования предиката «Сравнение»

Показать все дисциплины, в которых число учебных часов превышает 260:

```
SELECT electives.name, electives.hours FROM electives
WHERE electives.hours > 260;
```

Результат выполнения кода представлен на рис. 2.24.

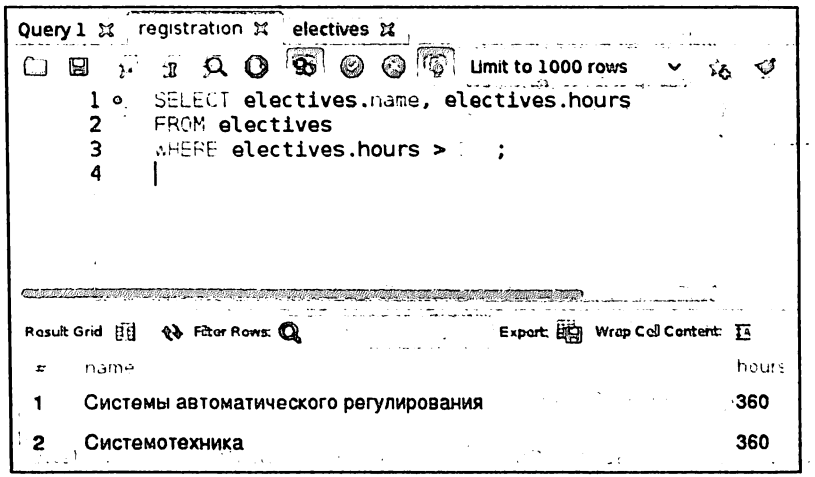


Рис. 2.24. Дисциплины, в которых число учебных часов превышает 260

Также предикаты могут быть построены с помощью логических операторов **AND**, **OR** или **NOT** и скобок, используемых для определения порядка вычисления выражения. Вычисление выражения выполняется по традиционным правилам:

- выражение вычисляется слева направо;
- первоначально вычисляются подвыражения в скобках;
- операторы **NOT** имеют приоритет над операторами **AND** и **OR**;
- операторы **AND** имеют приоритет над оператором **OR**.

Внимание! Для установления иного порядка вычисления следует использовать скобки.

Так, запрос, у каких дисциплин число учебных часов больше или равно 200, но меньше или равно 300, имеет следующий вид:

```
SELECT electives.name, electives.hours FROM electives
WHERE electives.hours>=200 And electives.hours<=300;
```

Аналогичным образом работает запрос из диапазона значений. В этом случае вместо знаков используется служебное слово **Between**. Если ищется величина не из заданного диапазона, то используется выражение **Not Between**.

Операторы IN и NOT IN

Рассмотрим пример запроса: определить, на какие дисциплины по выбору никто не зарегистрировался. Результат выполнения представлен на рис. 2.25.

```
SELECT name FROM electives
WHERE id_electives NOT IN ( SELECT id_electives from
registration);
```

1	Системотехника
2	Специальные разделы компьютерной графики
3	Стандартизация и сертификация программного обеспечения

Рис. 2.25. Поиск дисциплин, которые никто не выбрал

Соответствие шаблону

С помощью оператора **LIKE** можно выполнять сравнение выражения с заданным шаблоном, в котором допускается использование символов-заменителей:

- символ % — вместо этого символа может быть подставлено любое количество произвольных символов;
- символ _ заменяет один символ строки;
- [ ] — вместо символа строки будет подставлен один из возможных символов, указанный в этих ограничителях;
- [^] — вместо соответствующего символа строки будут подставлены все символы, кроме указанных в ограничителях.

Приведем еще несколько полезных примеров.

Найти студентов, в номере телефона которых третья цифра — 7.

```
SELECT student.last_name, student.phone FROM student
WHERE student.phone LIKE "__7%";
```

Результат выполнения кода представлен на рис. 2.26.

Далее найдем студентов, у которых в номере телефона вторая цифра — 2 или 7. Для этого воспользуемся поиском по шаблону с использованием функции **RLIKE**, для которой строится регулярное выражение (более подробную информацию о регулярных выражениях можно найти в соответствующей литературе).

1	Сидорова	777-23-34
2	Алмазова	147-88-35

Рис. 2.26. Поиск по шаблону телефона с третьей цифрой 7

```
SELECT student.last_name, student.phone FROM student
WHERE student.phone RLIKE '^[0-9]{1}[27]{1}';
```

Результат выполнения кода представлен на рис. 2.27.

1	Иванов	123-45-67
2	Петров	321-54-76
3	Сидорова	777-23-34

Рис. 2.27. Поиск по регулярному выражению

И еще один пример: найти телефоны студентов, у которых имя начинается на букву «И».

```
SELECT student.first_name, student.last_name, student.phone
FROM student
WHERE student.first_name LIKE 'И%';
```

1	Иван	Иванов	123-45-67
2	Инна	Сидорова	777-23-34

Рис. 2.28. Поиск телефонов студентов с именем на букву «И»

Результат выполнения кода представлен на рис. 2.28.

Значение NULL

Оператор **IS NULL** используется для сравнения текущего значения со значением **NULL** — специальным значением, указывающим на отсутствие любого значения. **NULL** отличается и от строки нулевой длины (пустой строки), знака пробела или ноля. Оператор **IS NOT NULL** используется для проверки присутствия значения в поле.

#	last_name	first_name
1	Серова	Анна

Рис. 2.29. Фамилия и имя студента, у которого отсутствует телефон

Однако часто в запросах, если неизвестно, является ли поле пустым (значение не внесено) или в поле явно внесено значение NULL, следует использовать не только оператор IS NULL, но и сравнивать поле с пустым значением. Рассмотрим пример.

Найти студентов, у которых нет телефона. Результат выполнения представлен на рис. 2.29.

```
SELECT student.last_name, student.first_name FROM student
WHERE student.phone IS NULL OR student.phone = '';
```

ORDER BY

В общем случае строки в результирующей таблице SQL-запроса никак не упорядочены. Для их сортировки используется команда **ORDER BY** в операторе **SELECT**. **ORDER BY** сортирует данные выходного набора в заданной последовательности. Сортировка может выполняться по нескольким полям, в этом случае они перечисляются за ключевыми словами **ORDER BY** через запятую. Способ сортировки задается ключевым словом, указываемым за названием поля, по которому выполняется сортировка. По умолчанию реализуется сортировка по возрастанию. Явно она задается ключевым словом **ASC**. Для выполнения сортировки по убыванию необходимо после имени поля, по которому она выполняется, указать ключевое слово **DESC**. Оператор **ORDER BY** позволяет упорядочить выбранные записи в порядке возрастания или убывания значений любого столбца или комбинации столбцов, независимо от того, присутствуют эти столбцы в таблице результата или нет. Оператор **ORDER BY** всегда должен быть последним элементом в операторе **SELECT**.

Вывести список кафедр в алфавитном порядке.

```
SELECT cafedra.name
FROM cafedra
ORDER BY cafedra.name;
```

Результат выполнения запроса представлен на рис. 2.30.

В операторе **ORDER BY** может быть указано и больше одного элемента.

Рассмотрим пример вывода списка студентов и названий кафедр, на которых они учатся. Названия кафедр упорядочить в алфавитном

1	Информационной безопасности
2	Моделирования информационных систем и сетей
3	Прикладной математики
4	Программной инженерии

Рис. 2.30. Сортировка названий фирм по алфавиту

порядке. Фамилии студентов внутри кафедр вывести в обратном порядке.

```
SELECT cafedra.name, student.last_name, student.first_name,
student.second_name
FROM cafedra INNER JOIN student ON cafedra.id_cafedra =
student.id_cafedra
ORDER BY cafedra.name, student.last_name DESC;
```

1	Информационной безопасности	Сидорова	Инна	Сергеевна
2	Информационной безопасности	Иванов	Иван	Иванович
3	Моделирования информационных систем и сетей	Алмазова	Ольга	Петровна
4	Прикладной математики	Серова	Анна	Анатольевна
5	Программной инженерии	Петров	Петр	Петрович
6	Программной инженерии	Говоров	Олег	Иванович

Рис. 2.31. Упорядочивание по алфавиту кафедр и фамилий студентов в обратном порядке

Результат выполнения запроса представлен на рис. 2.31.

Использование агрегатных (итоговых) функций и вычисляемых полей

В общем случае для создания **вычисляемого поля** в списке **SELECT** следует указать некоторое выражение языка SQL. В этих выражениях применяются арифметические операции сложения, вычитания, умножения и деления, а также встроенные функции языка SQL. При построении сложных выражений могут понадобиться скобки.

Стандарты SQL позволяют явным образом задавать имена столбцов результирующей таблицы, для чего применяется предложение **AS**.

С помощью **агрегатных функций** в рамках SQL-запроса можно получить ряд обобщенных статистических сведений о множестве ото-

бранных значений выходного набора. Для применения агрегатных функций предполагается предварительная операция группировки, при которой все множество кортежей отношения разбивается на группы, в которых собираются кортежи, имеющие одинаковые значения атрибутов, которые заданы в списке группировки. Пользователю доступны следующие основные агрегатные функции:

- **Count** (выражение) — количество строк или непустых значений полей, которые выбрал запрос;
- **Min/Max** (выражение) — определяют наименьшее или наибольшее из множества значений данного поля;
- **Avg** (выражение) — эта функция позволяет рассчитать среднее арифметическое значение выбранных значений данного поля;
- **Sum** (выражение) — вычисляет сумму выбранных значений данного поля.

Агрегатные функции используют имя поля как аргумент. Все эти функции оперируют со значениями в единственном столбце таблицы или с арифметическим выражением и возвращают единственное значение. Выражение может вычисляться и по значениям нескольких таблиц. С функциями SUM и AVG могут использоваться только числовые поля. С функциями COUNT, MAX и MIN могут использоваться как числовые, так и символьные поля. При использовании с символьными полями MAX и MIN будут транслировать их в эквивалент ASCII кода и обрабатывать в алфавитном порядке.

При вычислении результатов любых функций сначала исключаются все пустые значения, после чего требуемая операция применяется только к оставшимся конкретным значениям столбца. Вариант **COUNT(*)** — особый случай использования функции **COUNT**, его назначение состоит в подсчете всех строк в результирующей таблице, независимо от того, содержатся там пустые, дублирующиеся или любые другие значения.

Если до применения обобщающей функции необходимо исключить дублирующиеся значения, следует перед именем столбца в определении функции поместить ключевое слово **DISTINCT**. Оно не имеет смысла для функций **MIN** и **MAX**, однако его использование может повлиять на результаты выполнения функций **SUM** и **AVG**, поэтому необходимо заранее обдумать, должно ли оно присутствовать в каждом конкретном случае. Кроме того, ключевое слово **DISTINCT** может быть указано в любом запросе не более одного раза.

Следует помнить, что агрегатные функции могут использоваться только в списке предложения **SELECT** и в составе предложения **HAVING**. Во всех других случаях это недопустимо. Если список в операторе **SELECT** содержит агрегатные функции, а в тексте запроса отсутствует фраза **GROUP BY**, обеспечивающая объединение данных в группы, то ни один из элементов списка оператора **SELECT** не может включать каких-либо ссылок на поля, за исключением ситуации, когда поля выступают в качестве аргументов агрегатных функций.

Рассмотрим запрос: посчитать число студентов, записавшихся на каждую из дисциплин по выбору. Результат выполнения запроса представлен на рис. 2.32.

```
SELECT electives.name, COUNT(registration.id_electives) AS
count_id_electives
FROM electives INNER JOIN registration ON
electives.id_electives = registration.id_electives
GROUP BY electives.name;
```

1	Инженерная графика	2
2	Системы автоматического регулирования	2
3	Специальные языки программирования	1
4	Теория автоматического управления	2
5	Теория алгоритмических языков и методы трансляции	1

Рис. 2.32. Подсчет числа студентов, записавшихся на каждую из дисциплин

GROUP BY

Запрос, в котором присутствует **GROUP BY**, называется группирующим запросом, поскольку в нем группируются данные, полученные в результате выполнения операции **SELECT**, после чего для каждой отдельной группы создается единственная суммарная строка. При наличии в операторе **SELECT** предложения **GROUP BY** каждый элемент списка в предложении **SELECT** должен иметь единственное значение для всей группы. Оператор **SELECT** может включать только следующие типы элементов: имена полей, агрегатные функции, константы и выражения, включающие комбинации перечисленных выше элементов.

Все имена полей, приведенные в списке предложения **SELECT**, должны присутствовать и в **GROUP BY** — за исключением случаев,

когда имя столбца используется в агрегатной функции, однако в **GROUP BY** могут быть имена столбцов, отсутствующие в списке предложения **SELECT**.

Если совместно с **GROUP BY** используется предложение **WHERE**, то оно обрабатывается первым, а группированию подвергаются только те строки, которые удовлетворяют условию поиска.

HAVING

HAVING может использовать только аргументы, которые имеют одно значение на группу вывода. При помощи **HAVING** отражаются все предварительно сгруппированные посредством **GROUP BY** блоки данных, удовлетворяющие заданным в **HAVING** условиям. Остальные столбцы можно специфицировать только внутри спецификаций агрегатных функций **COUNT**, **SUM**, **AVG**, **MIN** и **MAX**, вычисляющих в данном случае некоторое агрегатное значение для всей группы строк. Результатом выполнения оператора HAVING является сгруппированная таблица, содержащая только те группы строк, для которых результат вычисления условия поиска есть TRUE.

Понятие подзапроса

Часто нет возможности получить требуемые данные при помощи только одного запроса. Такая ситуация бывает в случаях, когда в предложении **WHERE** появляется значение, с которым надо сравнивать, но которое заранее не было определено (т. е. должно быть вычислено до момента выполнения оператора **SELECT**). В таких случаях используются законченные операторы **SELECT**, внедренные в тело другого оператора **SELECT**. Указанные внедренные операторы называются «внутренние подзапросы». Внутренний подзапрос представляет собой также оператор **SELECT**, а его синтаксис соответствует тем же правилам, что и основного оператора **SELECT**. Внешний оператор **SELECT** использует результат выполнения внутреннего оператора для определения содержания окончательного результата всей операции. Внутренние запросы могут быть помещены непосредственно после оператора сравнения (=, <, >, <=, >=, <>) в предложения **WHERE** и **HAVING** внешнего оператора **SELECT**. Они называются «подзапросы» или «вложенные запросы». Кроме того, внутренние операторы **SELECT** могут применяться в операторах **INSERT**, **UPDATE** и **DELETE**.

Таким образом, подзапрос — это инструмент создания временной таблицы, содержимое которой извлекается и обрабатывается внешним оператором. Текст подзапроса должен быть заключен в скобки.

Существует два типа подзапросов:

- **скалярный** подзапрос возвращает единственное значение. В принципе, он может использоваться везде, где требуется указать единственное значение;
- **табличный** подзапрос возвращает множество значений, т. е. значения одного или нескольких столбцов таблицы, размещенные в более чем одной строке. Он возможен везде, где допускается наличие таблицы.

Рассмотрим подзапрос с группировкой. Предположим, что каждый студент к окончанию обучения должен набрать определенное число часов дисциплин по выбору. Необходимо вычислить общее число часов, которое набрал студент на настоящий момент. Результат выполнения запроса представлен на рис. 2.33.

```
SELECT student.last_name, student.first_name,
student.second_name,
SUM(electives.hours) AS sum_hours
FROM student INNER JOIN (electives INNER JOIN registration ON
electives.id_electives = registration.id_electives) ON
student.id_student = registration.id_student
GROUP BY student.last_name, student.first_name,
student.second_name;
```

#	last_name	first_name	second_name	sum_hours
1	Иванов	Иван	Иванович	864
2	Петров	Петр	Петрович	180
3	Серова	Анна	Анатольевна	504
4	Сидорова	Инна	Сергеевна	612

Рис. 2.33. Число часов, набранных студентом

Использование операций EXISTS и NOT EXISTS

Ключевые слова **EXISTS** и **NOT EXISTS** предназначены для использования только совместно с подзапросами. Результат их обработки представляет собой логическое значение **TRUE** или **FALSE**. Для ключевого слова **EXISTS** результат равен **TRUE** в том и только в том случае, если в возвращаемой подзапросом результирующей таблице присутствует хотя бы одна строка. Если результирующая таблица под-

запроса пуста, результатом обработки операции **EXISTS** будет значение **FALSE**. Для ключевого слова **NOT EXISTS** используются правила обработки, обратные по отношению к ключевому слову **EXISTS**. Поскольку по ключевым словам **EXISTS** и **NOT EXISTS** проверяется лишь наличие строк в результирующей таблице подзапроса, то эта таблица может содержать произвольное количество столбцов.

Дальнейшая работа с базой данных может выполняться непосредственно при помощи редактора запросов Workbench. Однако желательно, чтобы к базе данных был написан интерфейс, например, на языке PHP с использованием библиотеки разработки jQuery.

Учащимся предлагается выполнить практическую работу, основанную на изученном теоретическом материале. Выполнение практической работы предполагает использование **MySQL Workbench** через SQL Editor для создания тестовой базы данных и выполнение некоторых основных действий с ней.

Порядок выполнения практической работы

1. Ознакомиться с описанием.
2. Открыть рабочую область программы MySQL Workbench для создания базы данных.
3. Создать тестовую базу данных в соответствии с заданием, полученным у преподавателя.
4. Определить связи между таблицами.
5. Выбрать для столбцов требуемые типы данных.
6. Внести данные в таблицы не менее 5—6 записей в каждую таблицу.
7. Создать запрос на выборку из одной таблицы, из нескольких таблиц. Запустить запрос на выполнение. Сохранить запрос в выбранном файле.
8. Открыть файл с сохраненным запросом. Выполнить запрос.
9. Создать пять запросов, используя не менее двух таблиц в каждом с различными операторами сравнения, принадлежности к множеству, группировки, поиску по шаблону, упорядочиванию и пр.
10. Создать запрос, использующий одну или несколько агрегатных функций.
11. Сохранить базу данных и выйти из программы.
12. Подготовить отчет по практической работе, содержащий основные этапы создания базы данных, ход работы при создании запросов, результат их выполнения.

Контрольные вопросы

1. Что является первым этапом проектирования базы данных?
2. Что необходимо определить при создании таблицы?
3. Хорошо ли с точки зрения безопасности работать с СУБД под пользователем **root**?
4. Какие способы создания таблицы в MySQL Workbench вы знаете?
5. Какие основные пункты меню используются для создания таблиц базы данных, для составления запросов к базе данных?
6. Для чего служит оператор **SELECT**? Приведите примеры его использования.
7. Какие разделы оператора **SELECT** вы знаете? Что задается в разделе GROUP BY?
8. Расскажите, каким образом при помощи оператора **SELECT** осуществить выборку всех данных из таблицы. Каким образом избежать выборки дублирующих значений?
9. Какие предикаты для использования в предложении **WHERE** вы знаете? Приведите пример использования предиката сравнения.
10. Для чего используется оператор **LIKE** в СУБД MySQL? Какие еще операторы, выполняющие аналогичные функции, вы знаете?
11. Для чего служит раздел оператора **SELECT ORDER BY**?
12. Какое ключевое слово используется для сортировки записей по убыванию?
13. Что такое агрегатные функции? Какие агрегатные функции вы знаете?
14. Что можно вычислить при помощи агрегатной функции **Count**?
15. Расскажите об использовании операции **NOT IN**.

Лабораторная работа 3
ОСНОВЫ АДМИНИСТРИРОВАНИЯ СЕРВЕРА MARIADB

Цель: изучение основ администрирования сервера MariaDB.

Следует отметить, что администрирование сервера является одним из ключевых навыков работы с **MariaDB**. Администрирование позволяет устанавливать необходимый тип соединения, создавать резервные копии баз данных (backup), заводить новые учетные записи для создания новых пользователей и назначать им права, а также проводить мониторинг работы сервера и изменять его настройки. Заметим, что администрирование сервера **MariaDB** через **MySQL Workbench** полностью аналогично администрированию сервера **MySQL** (см., например, [1]). Кроме того, большая часть изменений настроек (по сравнению с настройками по умолчанию) должна выполняться пользователями только в том случае, если они полностью представляют последствия внесения таких изменений. В противном случае следует оставить настройки по умолчанию и ограничиться только созданием нового соединения, новых ролей и импортом-экспортом баз данных.

Настройки сервера предназначены для обеспечения требуемых свойств соединения в конкретном сеансе. Работа с настройками предполагает создание и управление соединением, а также администрирование и конфигурирование сервера.

Очевидно, что режим администрирования доступен только из-под **root**. Для создания нового соединения в основном окне входа необходимо нажать на знак «+» (рис. 3.1). После этого на экране появится окно настроек сервера, как показано на рис. 3.2.

Создадим требуемые настройки для работы с сервером. Назовем наше соединение **connection1**.

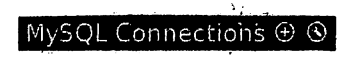


Рис. 3.1. Выбор знака «+» для создания нового соединения

Во вкладке выбора параметров (**Parameters**) выбирается протокол соединения (в данном случае стандартный протокол TCP/IP), производятся настройки

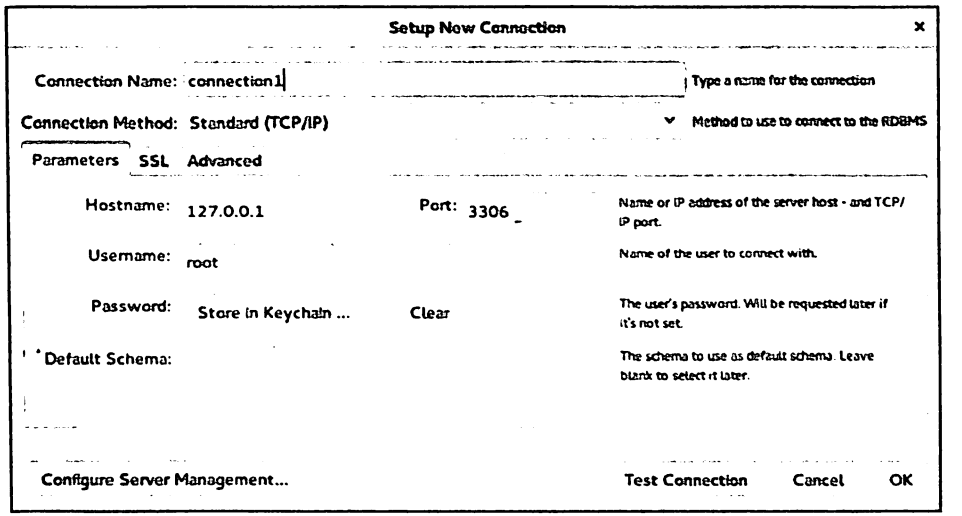


Рис. 3.2. Создание профиля настройки сервера для соединения

сервера и имени пользователя. Следует понимать, что начало работы происходит под именем **root** до тех пор, пока не созданы иные пользователи, имеющие право на работу с базой данных, и им не назначены определенные права.

Первоначально определяется хост — т. е. сервер, к которому будет обращаться клиентское приложение. В данном случае будем проводить настройки для локального сервера, а не для удаленного, поэтому выбираем опцию **localhost** (127.0.0.1).

Во вкладке **SSL** (рис. 3.3), как следует из ее названия, производятся настройки, связанные с протоколом **SSL**.

Во вкладке **Advanced** можно произвести настройки, связанные, например, с протоколом сжатия, или более сложные настройки аутентификации пользователей (рис. 3.4). Как было сказано ранее, рядовым пользователям не рекомендуется в данной вкладке проводить настройки, поскольку желательно точно знать, к каким последствиям приведет выбор тех или иных настроек. Если нет уверенности в правильности выбора, то следует оставлять настройки по умолчанию.

После того как настройки в этих вкладках будут созданы, можно проверить тестовое соединение с базой данных. Для этого необходимо ввести пароль (рис. 3.5).

Как было сказано выше, первоначально настройки проводятся с правами администратора, т. е. из-под **root**, для чего необходимо вве-

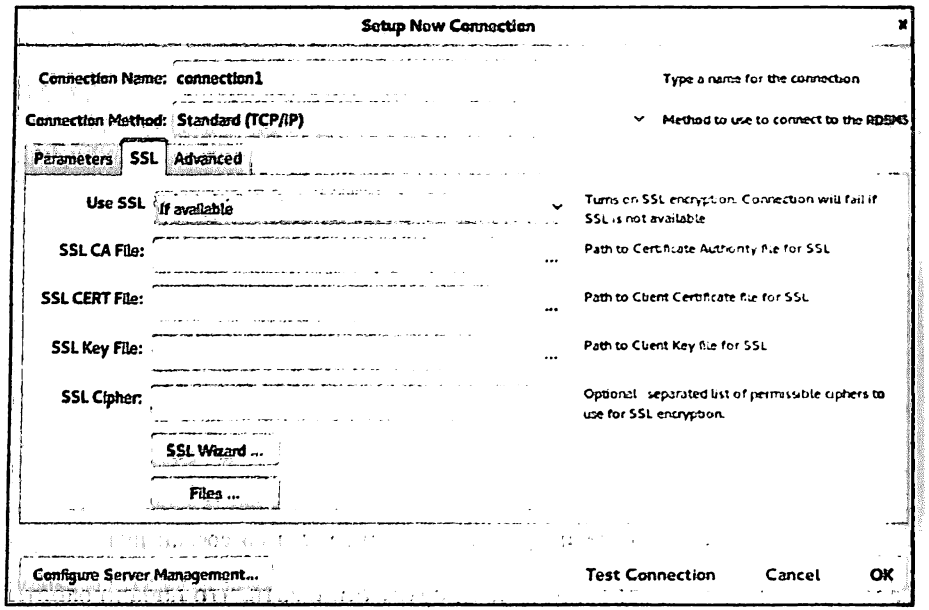


Рис. 3.3. Настройки протокола SSL

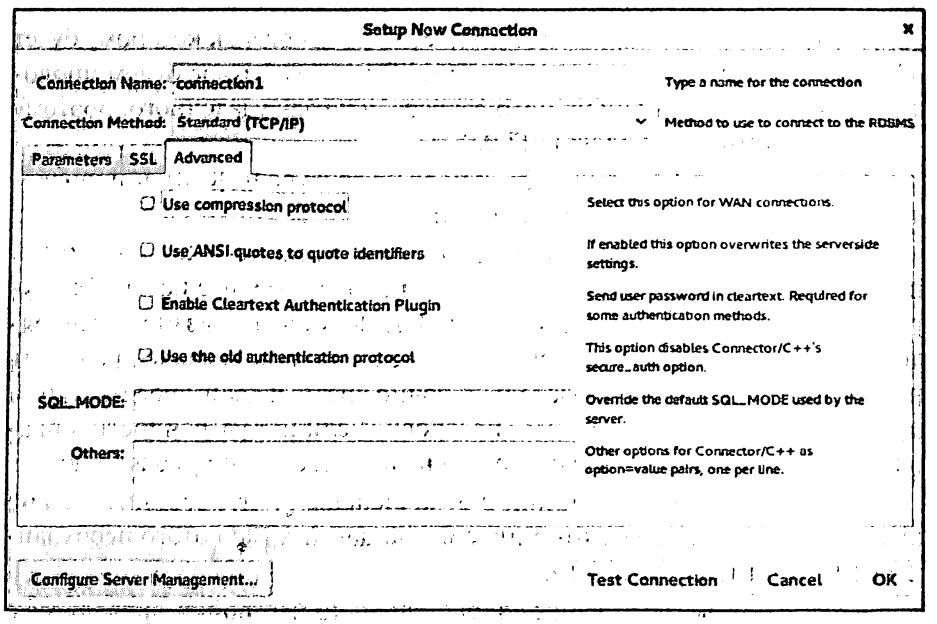


Рис. 3.4. Настройки вкладки **Advanced**

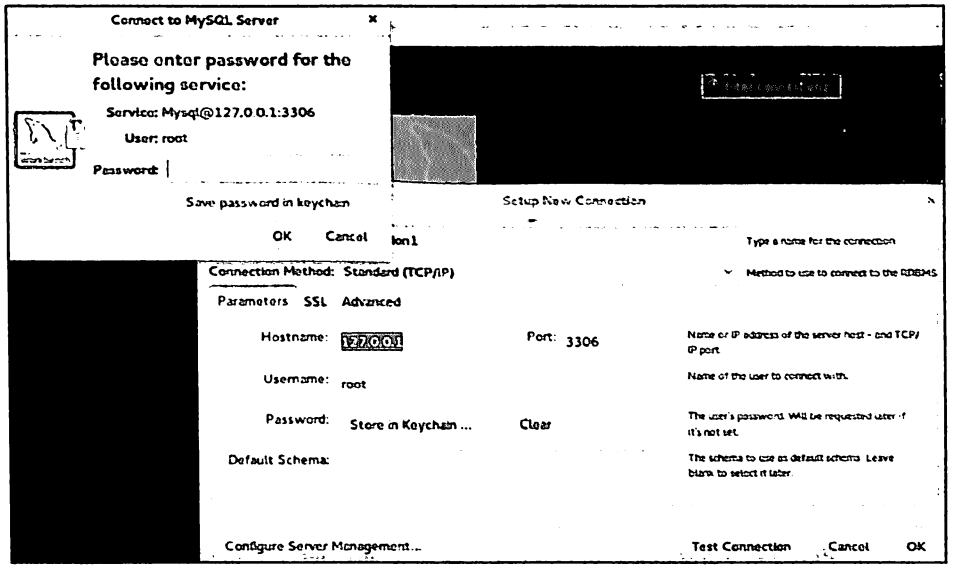


Рис. 3.5. Тестовое соединение с базой данных. Ввод пароля

сти пароль **root**. Если соединение установлено, то в открывшемся окне появится соответствующее сообщение. Если проверка не пройдена, необходимо вернуться на предыдущий шаг и проверить корректность установок.

Если пользователя на этапе создания соединения не устраивают настройки по умолчанию, можно перейти во вкладку установки конфигурации (позже это сделать не получится), однако настоятельно рекомендуется для пользователей среднего уровня оставить настройки по умолчанию. Для установки настроек необходимо выбрать внизу экрана кнопку **Configure Server Management** (см. рис. 3.3). На экране появится окно (рис. 3.6), в котором слева расположены пункты настроек. Переход к следующему пункту осуществляется по нажатию кнопки **Next**. В тех случаях, когда необходим возврат назад, используется кнопка **Back**, для отмены настройки — кнопка **Cancel**. Более подробное описание настроек можно посмотреть в [1]. Ниже коротко представлено описание основных этапов.

После пункта **Introduction** идет пункт тестового соединения с базой данных (**Test DB Connection**). Далее следуют пункты меню выбора операционной системы из списка. Если был выбран удаленный сервер, то следующим шагом будет **SSH Configuration** для установки безопасного удаленного соединения. Но поскольку в данном случае

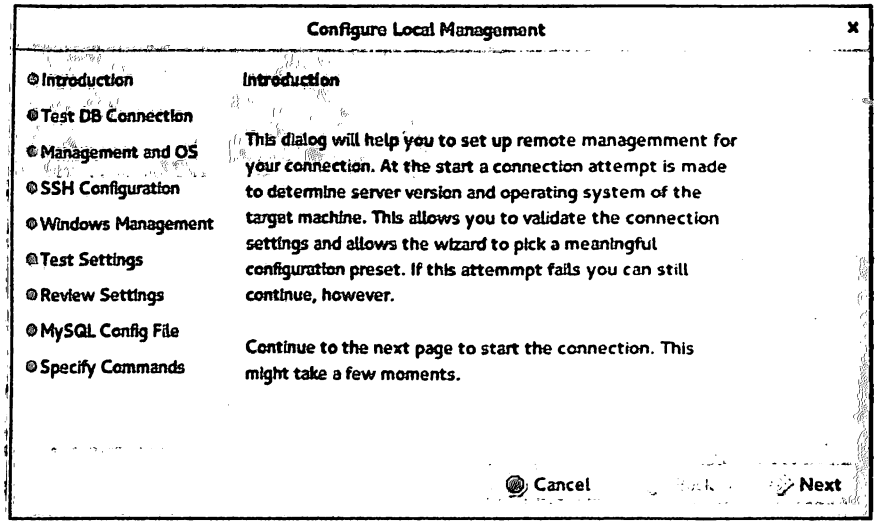


Рис. 3.6. Установка конфигураций **Configure Server Management**

работа идет с локальным сервером, то следующим шагом будет проверка настроек — **Test Settings**.

При появлении проблем с настройками по нажатии кнопки **Show Logs** можно получить подробное описание проблемы. В том случае, если после проверки установок ошибки не были обнаружены, осуществляется переход на шаг **Review Settings**. Если необходимо еще раз проверить установки, следует нажать кнопку **I'd like to review the settings again**. Если нет, то следует нажать кнопку **Continue**. В случае если нажата кнопка **Review**, все настройки будут отображены на странице в виде отчета. Просмотр настроек в обобщенном виде позволяет понять, есть ли необходимость вернуться к предыдущим шагам и внести исправления. **Specify Commands** позволяет создать настройки для запуска, останова и проверки статуса текущего соединения. Желательно, чтобы без необходимости эти настройки не изменялись недостаточно квалифицированными пользователями и оставались настроенными по умолчанию. Далее осуществляется переход к завершению настроек **Complete Setup**. На этой странице присваивается имя настроенному типу соединения.

После завершения всех действий следует нажать кнопку **Finish**, и настройки будут созданы. В этом случае в основном окне программы появится соединение, для которого произведены настройки.

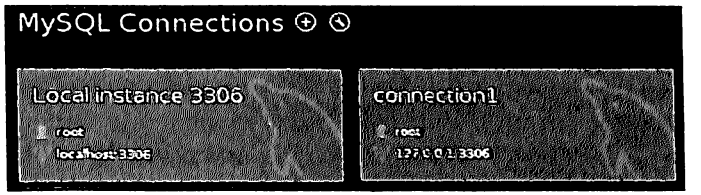


Рис. 3.7. Созданное соединение **connection1**

После этого соединение будет создано, и с ним можно работать (рис. 3.7).

Аналогичным образом, после того как будут созданы пользователи (об этом будет сказано ниже), можно создать соединение для любого пользователя.

Для вызова контекстного меню соединения необходимо щелкнуть правой кнопкой мыши на соответствующей области, в которой описывается соединение (рис. 3.8). Контекстное меню включает в себя стандартные пункты, связанные с открытием, редактированием, удалением, перемещением и некоторыми другими действиями с соединением.

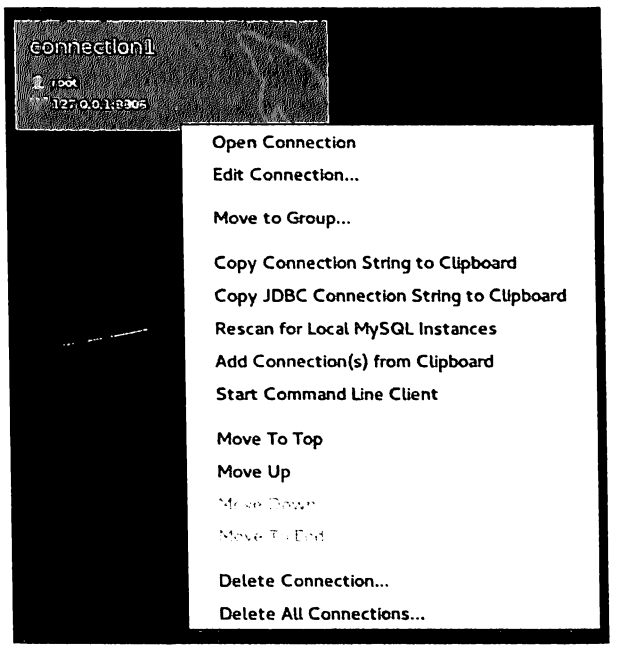


Рис. 3.8. Контекстное меню соединения **connection1**

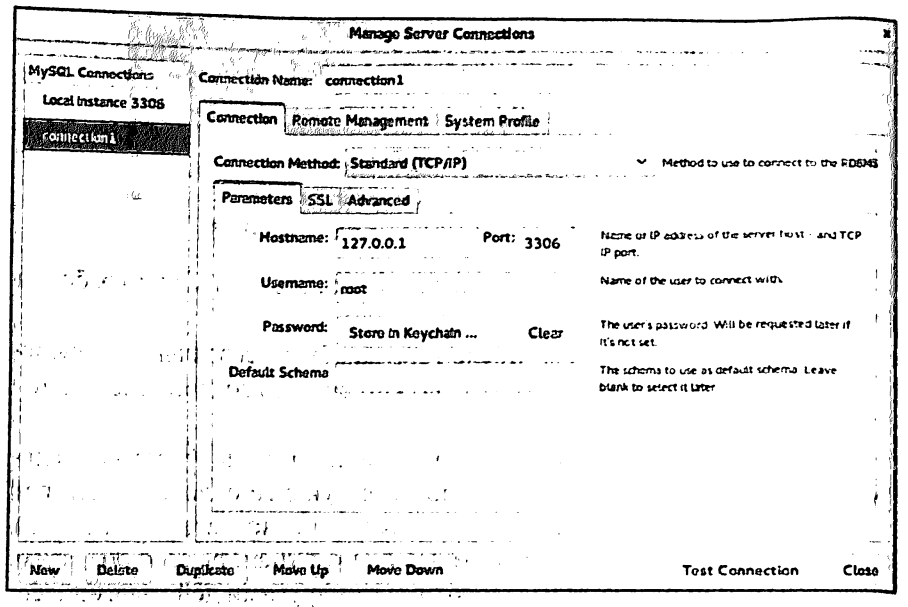


Рис. 3.9. Основные параметры соединения

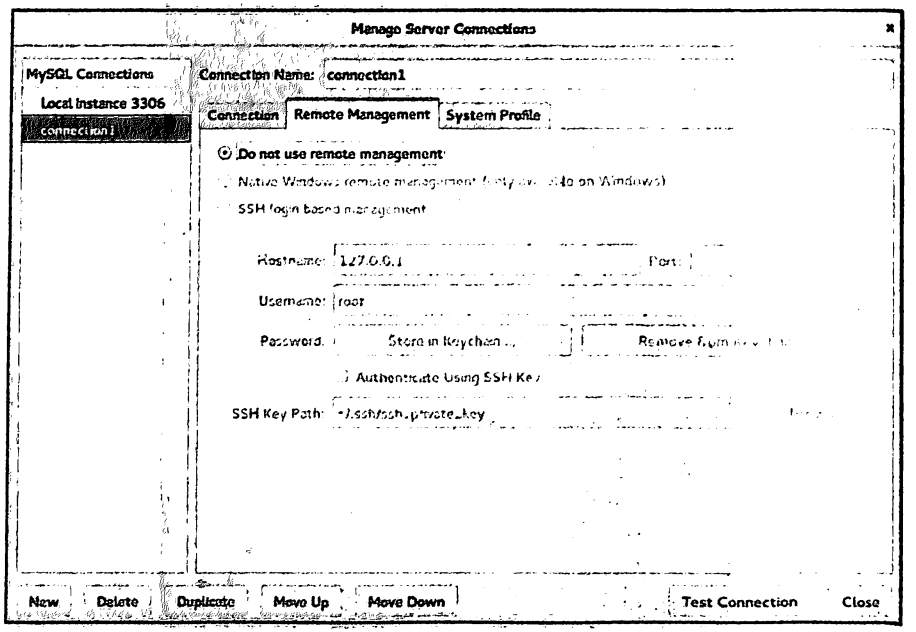


Рис. 3.10. Настройка удаленного соединения

После того как соединение создано, его можно редактировать. Если вызвать пункт меню **Edit Connection** (рис. 3.9), то появится окно для настройки соединения с тремя вкладками. Первая вкладка **Connection** предназначена для задания параметров соединения (рис. 3.9) и полностью аналогична рассмотренной выше вкладке на рис. 3.2.

Вкладка **Remote Management** позволяет настроить параметры удаленного соединения (рис. 3.10).

Третья вкладка **System Profile** предназначена для задания настроек (в том числе типа) операционной системы (рис. 3.11).

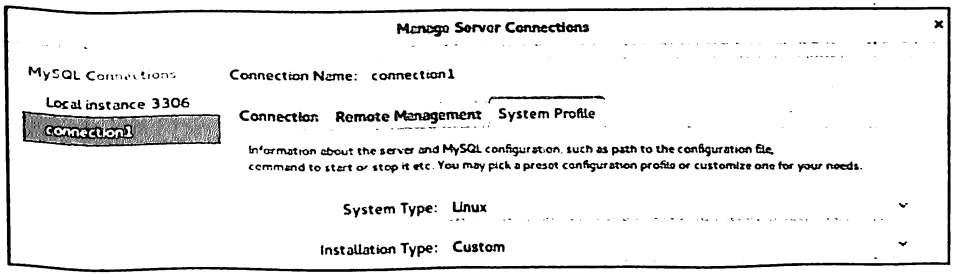


Рис. 3.11. Настройка операционной системы

Далее это соединение можно вызывать и работать с теми настройками, которые были установлены ранее. Для вызова соединения можно щелкнуть по нему мышью или выбрать пункт меню **Open Connection**. Откроется вкладка конкретного соединения (рис. 3.12).

В верхней части находится привычное меню, а слева — меню для администрирования работы сервера: **Management**, **Instance** и **Performance**. Вкладка **Management** предназначена для управления соединением. Первый пункт меню **Server Status** содержит информацию о соединении.

Перейдем на вкладку **Administration Server Status** (рис. 3.13) и рассмотрим подробнее ее области. Слева, как было сказано выше, находятся вкладки для администрирования сервера.

В центре расположены все данные о настройках параметров соединения: хост, порт и текущая версия сервера (рис. 3.14), о базовых директориях, а также имеющемся месте на диске, включенных/выключенных логах, режиме аутентификации (рис. 3.15).

Слева находится область, в которой можно посмотреть статус и режим загрузки сервера (рис. 3.16).

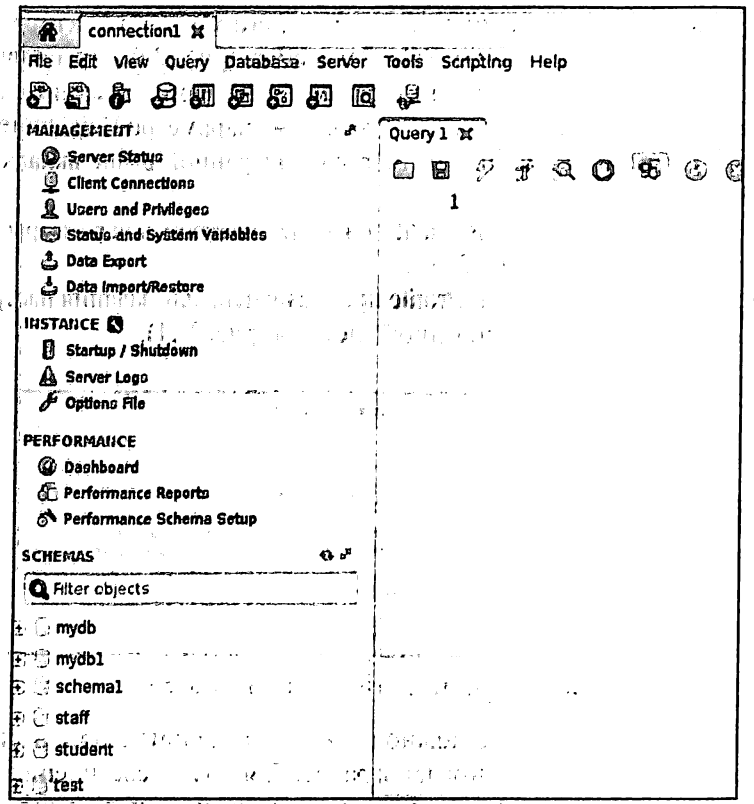


Рис. 3.12. Вкладка соединения **connection1**

Рассмотрим подробнее остальные пункты меню для администрирования сервера.

Пункт меню **Client Connection** предназначен для выдачи информации о пользователях и соединениях (рис. 3.17).

Далее следует пункт меню **Users and Privileges** (рис. 3.18), который предназначен для создания нового пользователя, назначения ему ролей и привилегий, а также для редактирования полномочий уже имеющихся пользователей либо для их удаления.

В данном окне существуют дополнительные вкладки (рис. 3.19), которые могут быть настроены для конкретного пользователя, о чем будет сказано ниже. Первая вкладка **Login** предназначена для создания новой учетной записи (нового пользователя), изменения или удаления имеющихся учетных записей. **Account Limits** позволяет настроить максимальное количество запросов, изменений, соединений и пр.

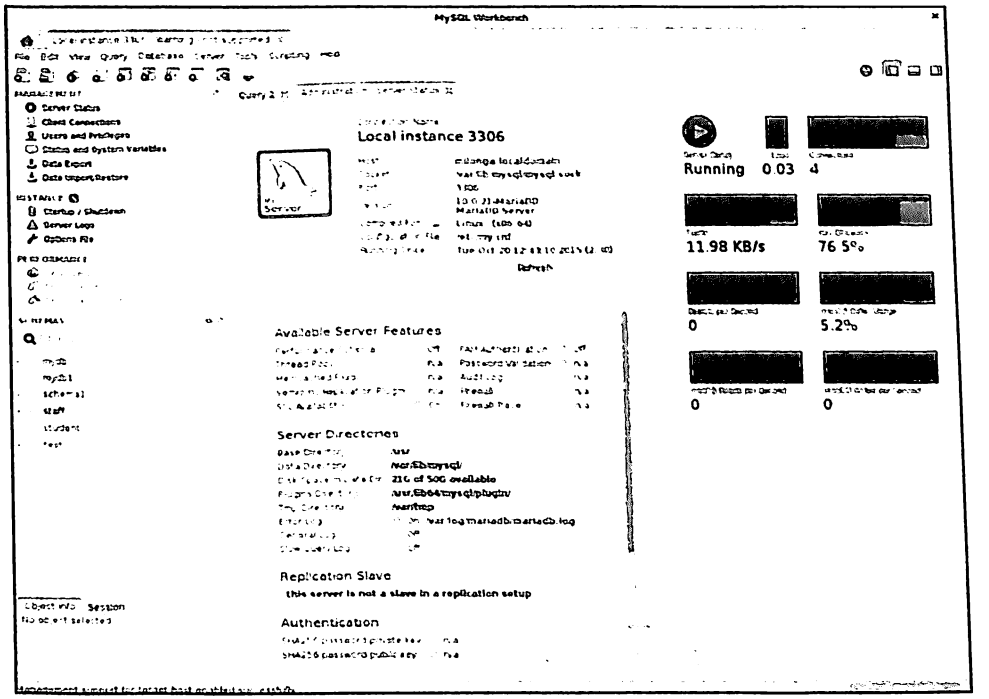


Рис. 3.13. Вкладка **Administration Server Status**

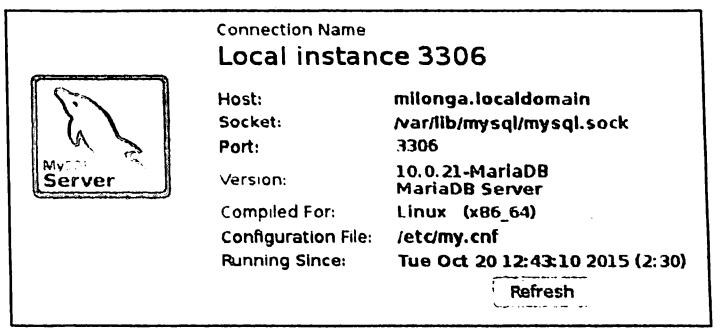


Рис. 3.14. Настроенные параметры соединения

в течение часа для конкретного пользователя. **Administration Roles** предназначена для назначения роли (ролей) пользователю. Вкладка **Schema Privileges** позволяет назначать привилегии пользователям. Именно в ней следует производить все настройки после того, как будет создан новый пользователь. Следует также заметить, что все эти

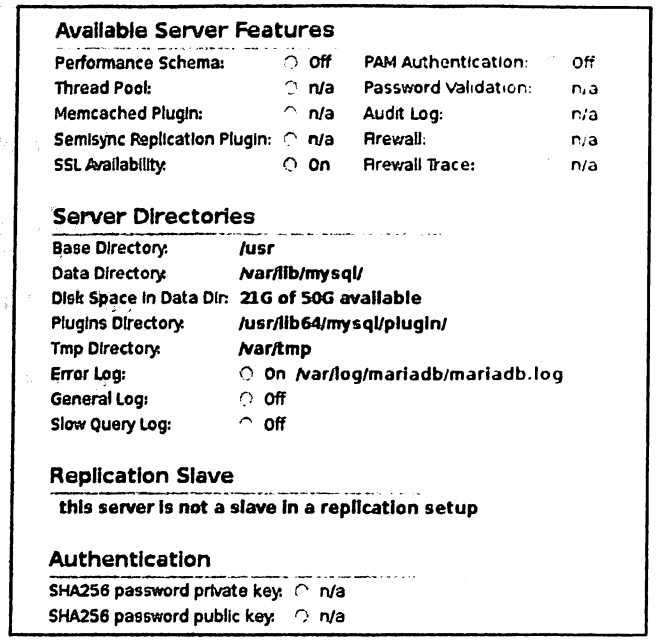


Рис. 3.15. Настроенные базовые директории и логи

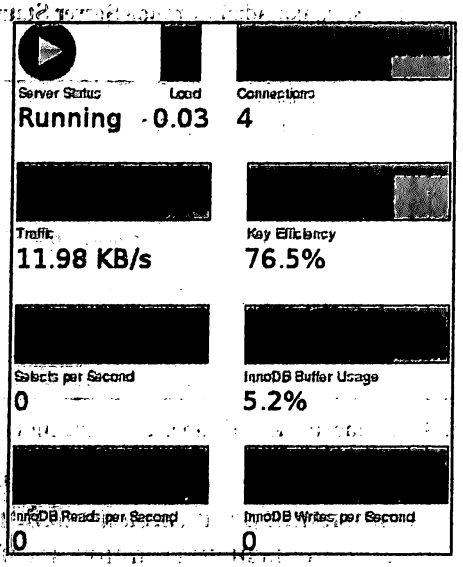


Рис. 3.16. Статус и режим загрузки сервера

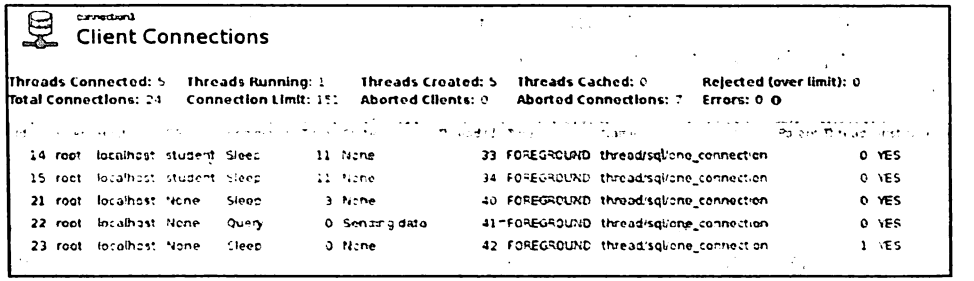


Рис. 3.17. Статус и режим клиентских соединений

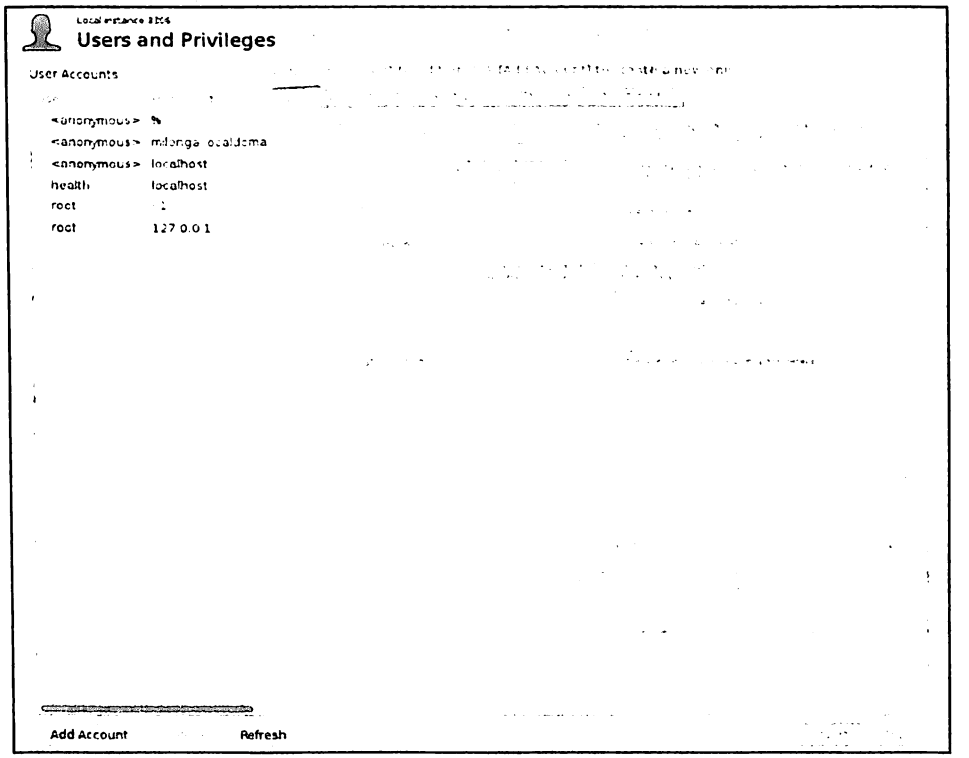


Рис. 3.18. Окно **Users and Privileges**

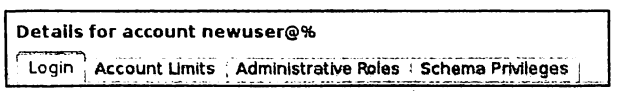


Рис. 3.19. Вкладки окна **Users and Privileges**

действия могут быть выполнены только из-под **root** — режима администратора (или пользователя, обладающего соответствующей привилегией).

Далее рассмотрим процесс создания нового пользователя — новой учетной записи. Выбираем кнопку внизу окна **Add Account**. В открывшейся вкладке необходимо задать имя пользователя, пароль, подтверждение пароля, а также выбрать сервер. В качестве сервера может выступать локальный сервер (**localhost**), удаленный сервер (для удаленного сервера рекомендуется указывать IP-адрес, а не имя) или, если в этом поле указано значение %, то доступ может производиться с любого сервера. Если пароль по мнению сервера будет легко взломать, то появится надпись «**Weak password**» и пароль необходимо будет изменить (рис. 3.20).

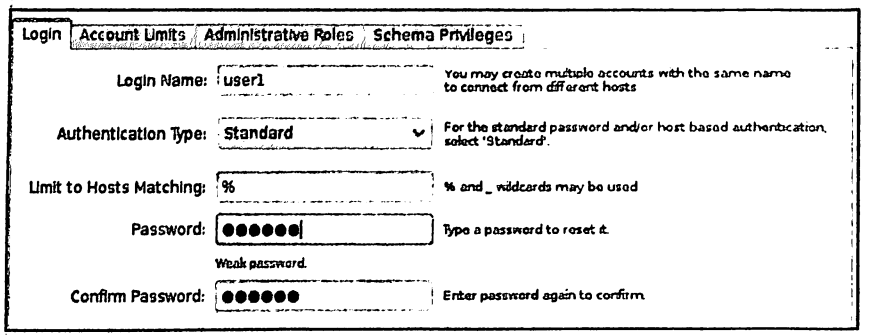


Рис. 3.20. Создание нового пользователя

После заполнения полей новый пользователь (в данном случае с именем **user1**) будет создан (рис. 3.21).

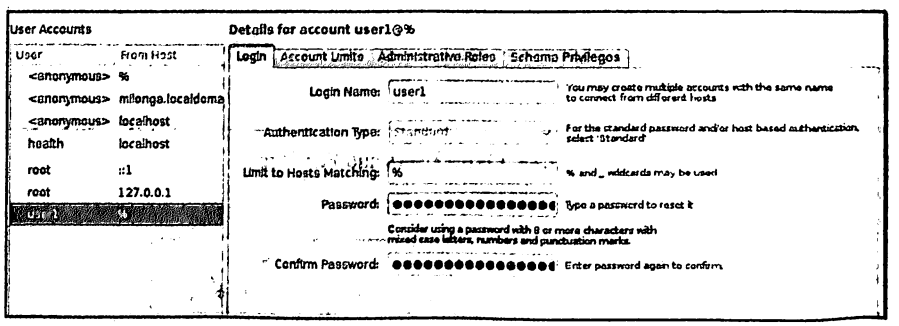


Рис. 3.21. Новый пользователь **user1** создан

Проведем дальнейшие настройки во вкладке **Account Limits** (рис. 3.22). Зададим все необходимые параметры ограничения. Рядом с полем, для которого можно произвести настройки, расположена поясняющая надпись.

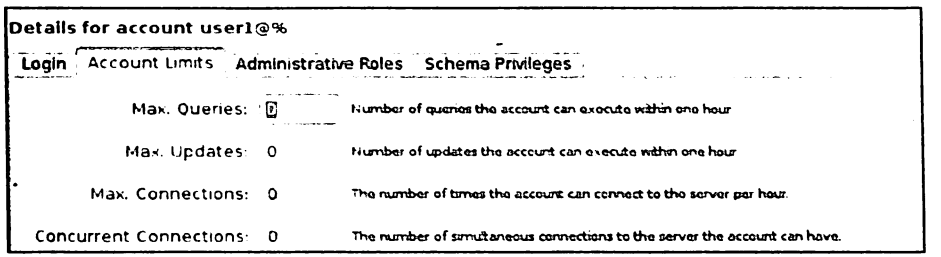


Рис. 3.22. Настройки во вкладке **Account Limits**

Далее можно настроить административные роли, проставив галочку напротив соответствующей опции во вкладке **Administrative Role** (рис. 3.23). Это наиболее простой способ назначения определенных прав созданному пользователю. Пользователю может быть назначено несколько ролей в зависимости от числа выбранных и отмеченных галочками. Например, если пользователю назначена роль **BackupAdmin**, то пользователь получает возможность работать с командами **EVENT**, **LOCK TABLES**, **SELECT**, **SHOW DATABASES**.

Возможный выбор ролей включает:

- **DBA**: имеет все права;
- **MaintenanceAdmin**: права поддержки сервера;
- **ProcessAdmin**: права мониторинга и сброса процесса пользователя;
- **UserAdmin**: права создания пользователя и переустановки пароля;
- **SecurityAdmin**: права по управлению логинами и сервером;
- **MonitorAdmin**: права контроля сервера;
- **DBManager**: права контроля баз данных;
- **DBDesigner**: права для создания и обратного инжиниринга любой схемы базы данных;
- **ReplicationAdmin**: права для установки и управления копированием;
- **BackupAdmin**: права, необходимые для резервного копирования баз данных.

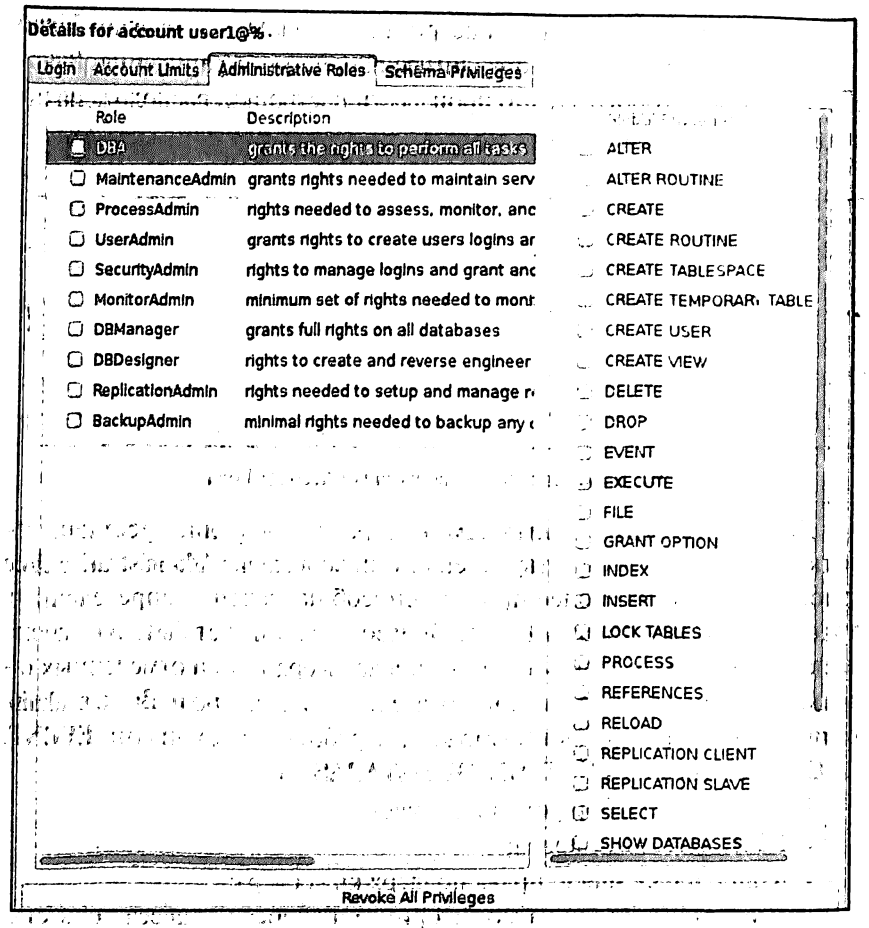


Рис. 3.23. Вкладка **Administrative Role**

После того как настройки учетной записи произведены, можно настроить привилегии через вкладку **Schema Privileges** (рис. 3.24). Для этого необходимо слева в окне **User Account** навести курсор на пользователя, для которого производятся настройки, выделить его (надпись о том, что настройки производятся для пользователя **user1**, появится вверху) и нажать на кнопку **Add Entry**. Обратите внимание, что на данном этапе опции для настроек неактивны. Это связано с тем, что пока еще не осуществлен выбор сервера и базы данных.

Выберем базу данных из списка, например **test** (рис. 3.25). После этого опции для настроек становятся активными (рис. 3.26). Выберем кнопку **Select All**, галочки автоматически будут поставлены у всех опций (рис. 3.27). Если необходимо разрешить не все опции, то соответственно галочки устанавливаются для разрешенных опций.

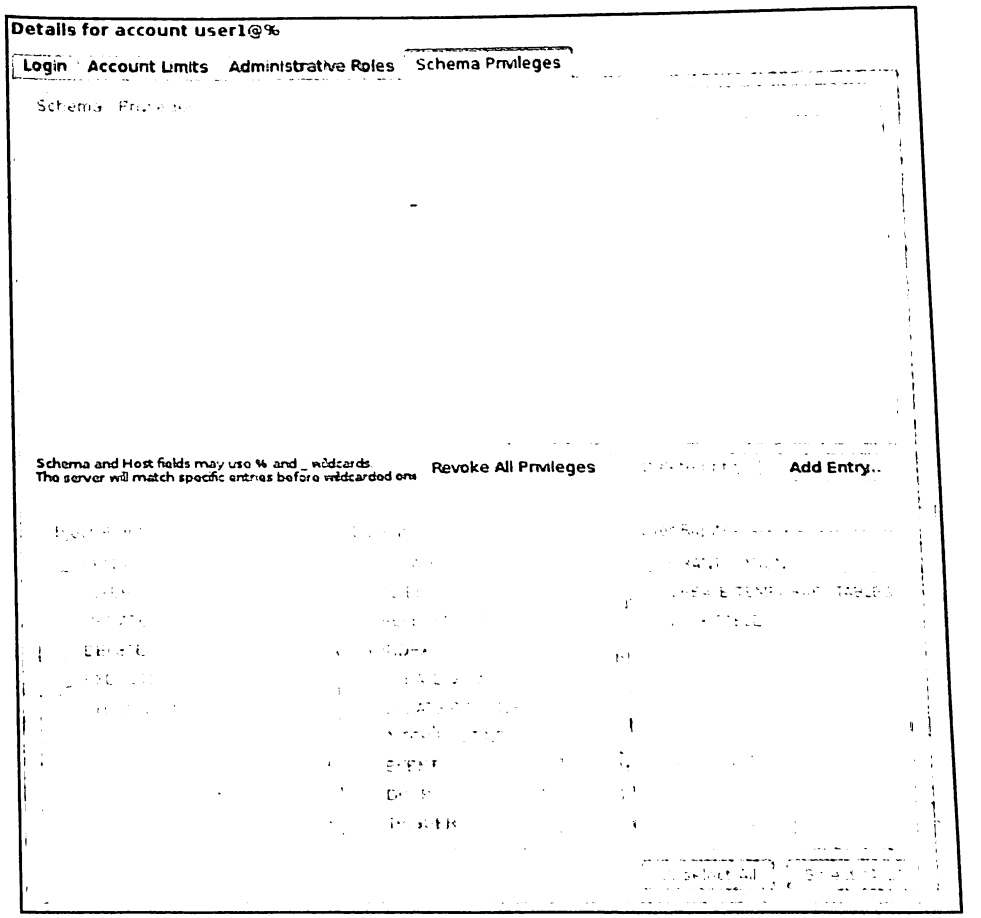


Рис. 3.24. Настройка привилегий через вкладку **Schema Privileges**

Если необходимо отменить привилегии, то выбирается кнопка **Revoke All Privileges**, после чего на экране появляется модальное окно с требованием подтверждения (рис. 3.28).

Итак, процесс создания нового пользователя завершен. Следует также заметить, что создание нового пользователя будет отражено при дальнейшей работе с сервером. Например, при создании ER-диаграмм, перейдя на вкладку **Users**, можно будет произвести дальнейшие настройки для появившегося там вновь созданного пользователя (см. лабораторную работу 5).

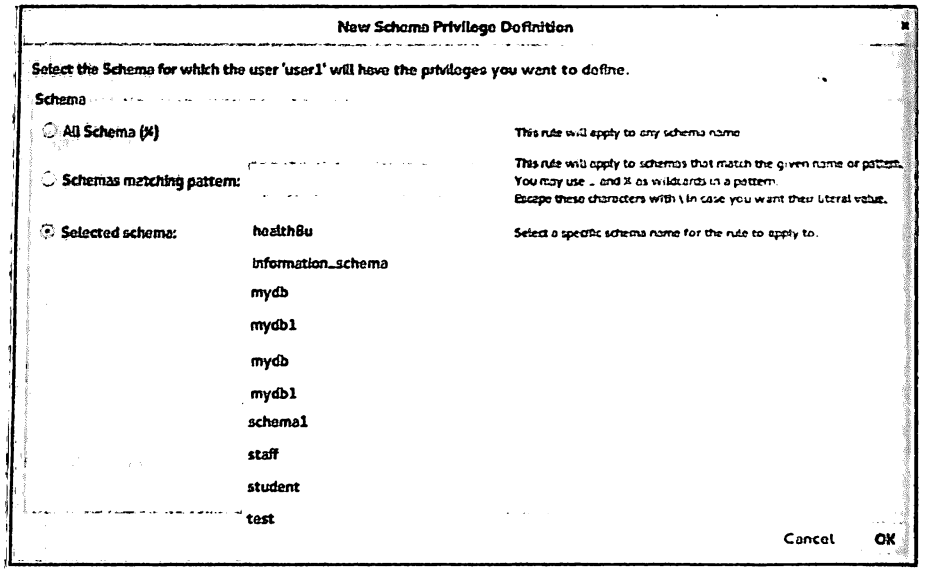


Рис. 3.25. Выбор базы данных

Вернемся к меню **Management** и рассмотрим его остальные пункты.

Следующим пунктом является **Status and System Variables**. В открывающемся окне при вызове данного пункта меню можно посмотреть информацию о переменных состояния и системы в соответствующей внутренней вкладке (рис. 3.29). В данном окне настройки не производятся. Служит для мониторинга состояния системы.

Следующий пункт меню — **Data Export** — служит для экспорта данных (рис. 3.30). В нем есть две вкладки: выбора объектов для экспорта (**Object Section**) и наблюдения за ходом экспорта (**Export Progress**).

Для экспорта данных в окне слева **Tables to Export** (выбор базы данных для экспорта) необходимо выбрать базу данных (рис. 3.31).

Когда она будет выбрана, необходимо выбрать таблицу (таблицы) в окне слева (рис. 3.32).

Далее в нижней части экрана переходим в область **Export Options** и выбираем **Export to Self-Contained File** (рис. 3.33). Отмечаем необходимость создания базы (**Include Create Shema**), в противном случае при открытии на другом компьютере базу данных придется создавать вручную.

Теперь необходимо выбрать место, куда будет сохранен файл. Для этого в пункте меню **Export to Self-Contained File** выбираем директорию и задаем название файла (рис. 3.34).

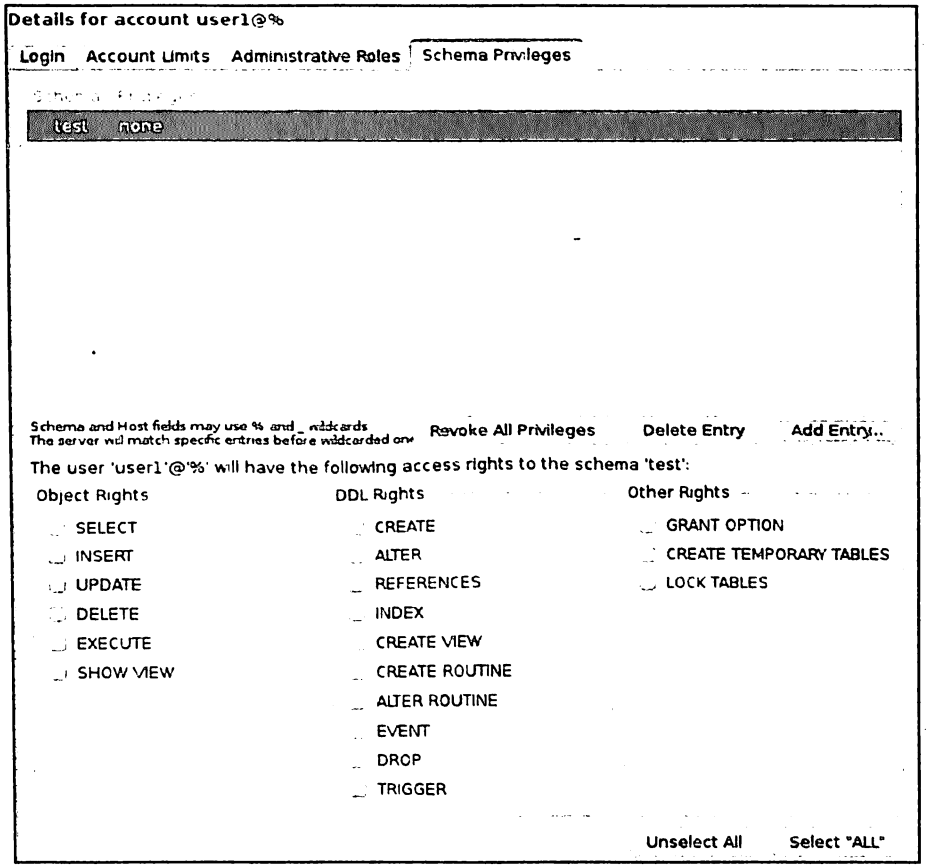


Рис. 3.26. Окно для определения привилегий

Нажав кнопку **Start Export**, запускаем экспорт файла в заранее определенное место на диске (см. рис. 3.33). После этого во вкладке **Export Progress** в поле **Log** будут показаны данные, в какую директорию осуществлен экспорт (рис. 3.35).

Если открыть файл, в который был осуществлен экспорт, например, сделать это через **Krusader** (рис. 3.36, 3.37), то будет видно, что файл содержит текст на языке **SQL** для создания базы данных и таблицы (таблиц). Следует помнить, что имеется единственный способ изменить имя базы данных — сделать это в данном файле вручную. Средствами же **MySQL Workbench** это сделать нельзя.

Далее рассмотрим импорт. Для выполнения импорта следует перейти на соответствующий пункт меню (рис. 3.38). В нем также есть

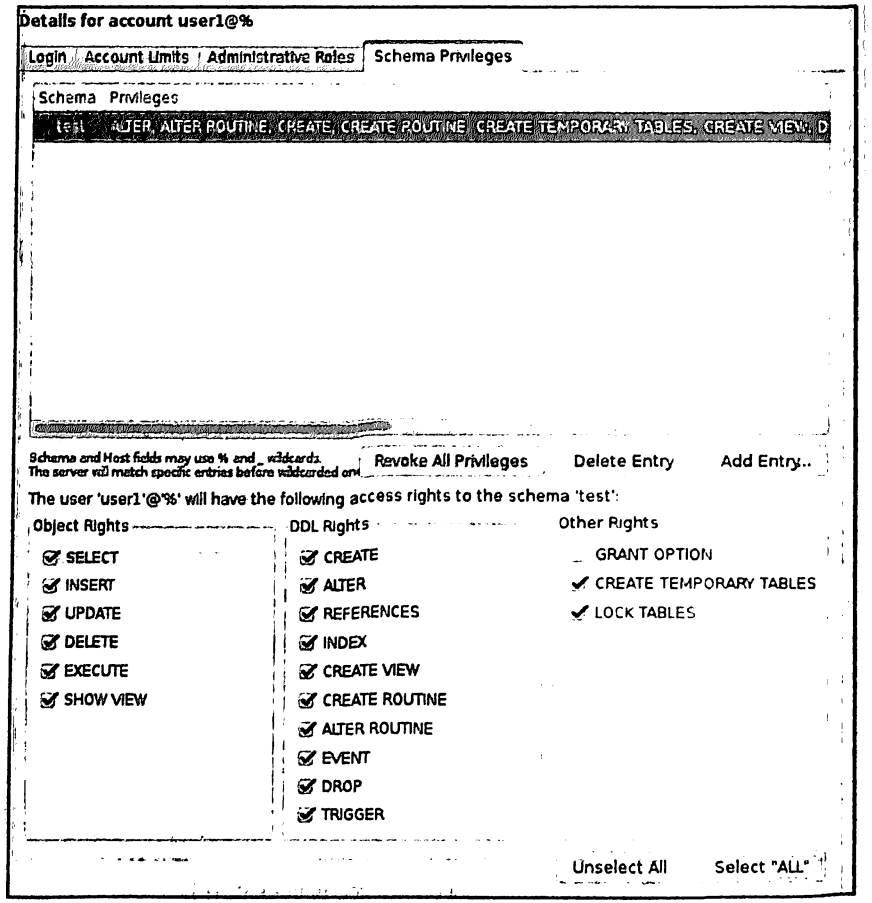


Рис. 3.27. Выбраны все привилегии

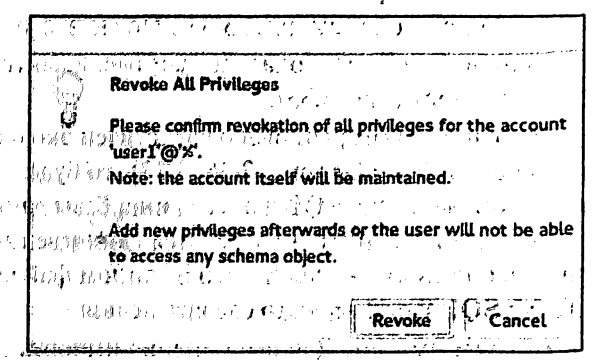


Рис. 3.28. Подтверждение отмены привилегий

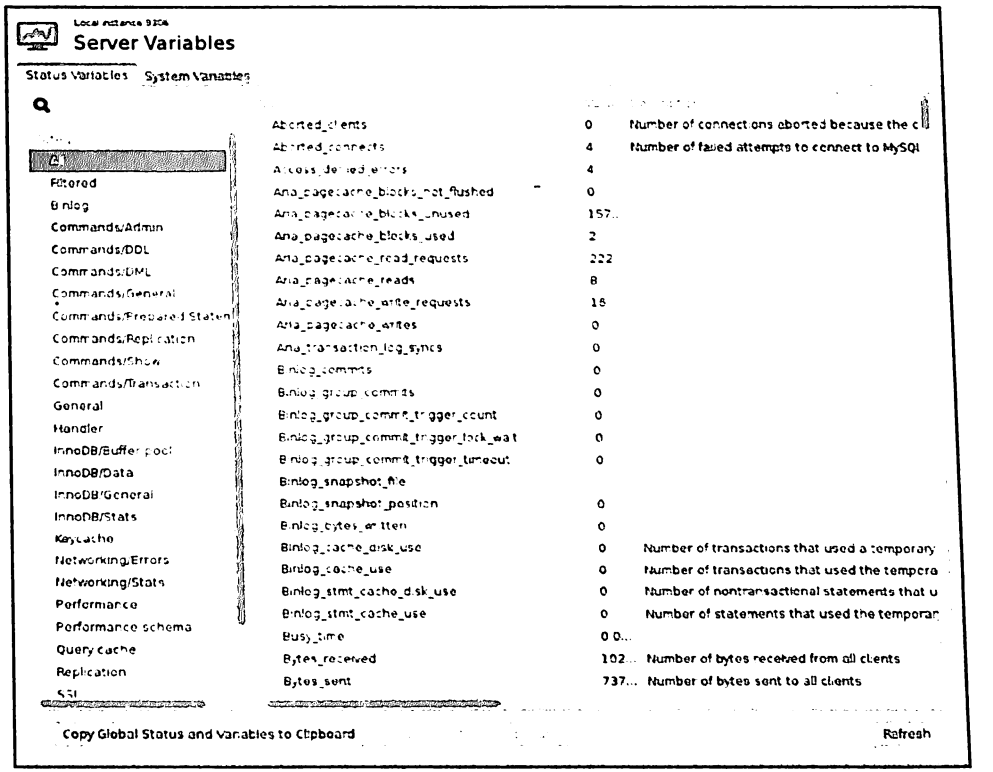


Рис. 3.29. Окно **Status and System Variables**

две вкладки: выбора импорта с диска (**Import From Disk**) и наблюдения за ходом импорта (**Import Progress**).

В поле **Import From Disk** выбираем опцию **Import to Self-Contained File** (рис. 3.39), указываем путь к файлу и нажимаем на кнопку **Start Import**.

Во вкладке **Import Progress** в поле **Log** будут показаны данные, откуда осуществлен импорт (рис. 3.40).

Следующая вкладка **Instance** предназначена для настройки параметров сервера.

Ниже идет вкладка **Performance**, которая позволяет получить статистические данные (пункт меню **Performance Dashboard**) и отчеты (пункт меню **Performance Reports**) по производительности. **Performance Schema Setup** содержит настройку для получения статистики о производительности сервера.

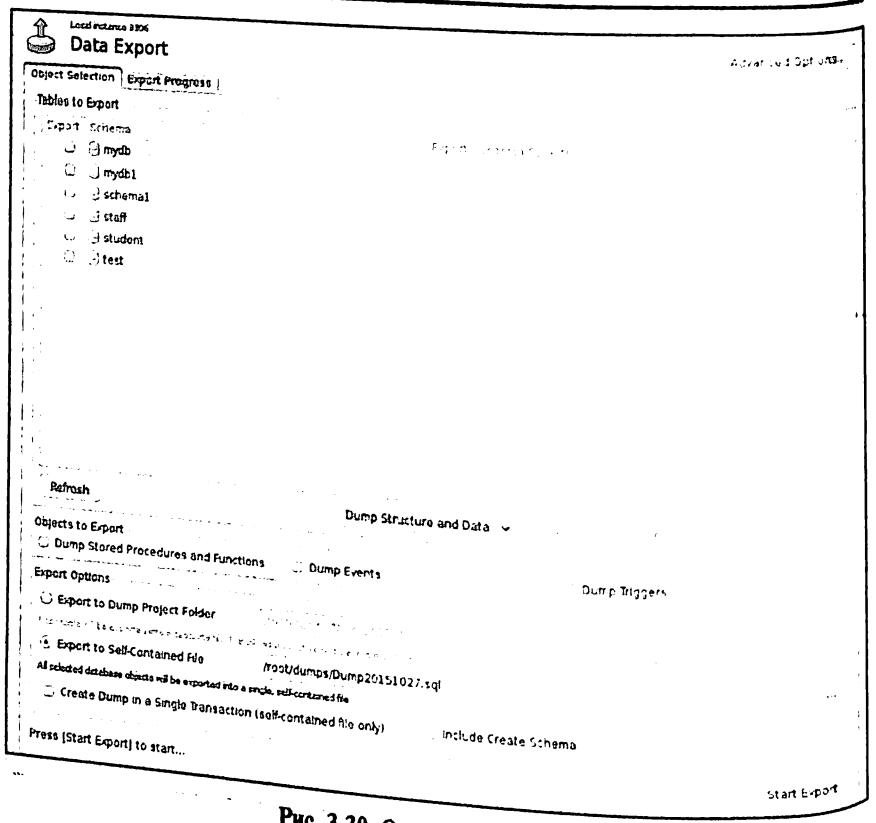


Рис. 3.30. Окно **Data Export**

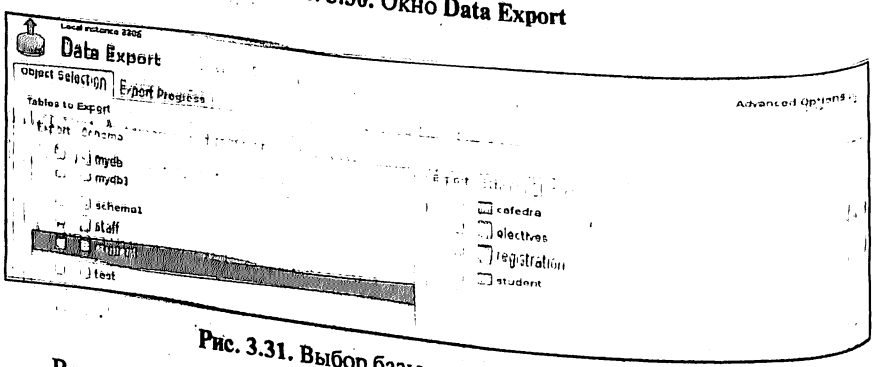


Рис. 3.31. Выбор базы данных для экспорта

Вернемся к вкладке **Instance**. Первый пункт меню **Startup/Shutdown** служит для запуска/остановки сервера. В поле **Startup Message Log** размещена информация о состоянии сервера (рис. 3.41).

При вызове пункта меню **Server Logs** открывается окно с системными сообщениями (рис. 3.42).

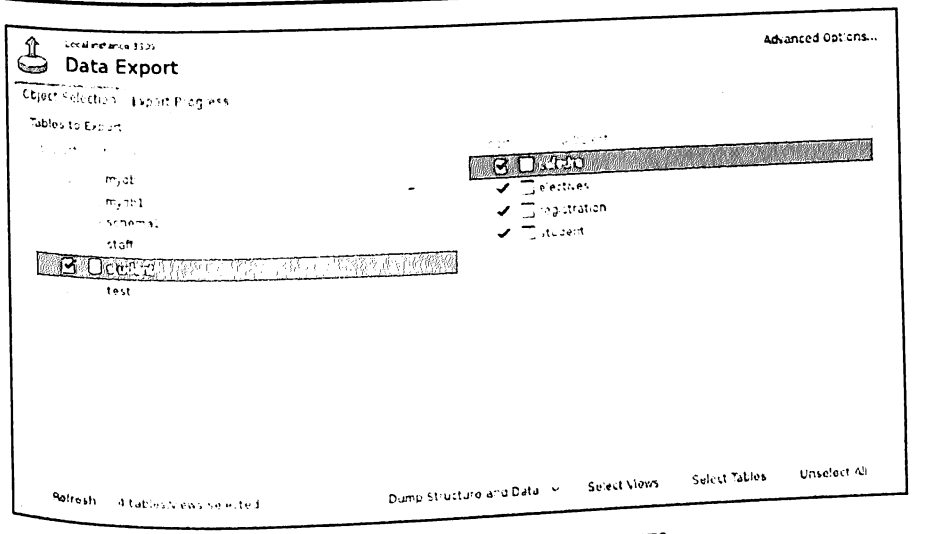


Рис. 3.32. Выбор таблиц для экспорта

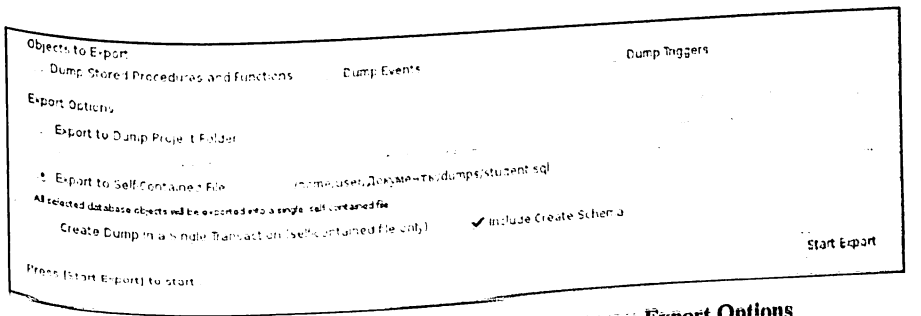


Рис. 3.33. Выбор опций для экспорта в области **Export Options**

Для более тонкой настройки соединения необходимо вызвать пункт меню **Options File** (рис. 3.43).

При выборе пункта меню **Options File** отображается вкладка **General** (общие настройки). Как было сказано ранее, следует производить настройки только в случае, если в этом есть реальная необходимость и пользователь понимает результат своих действий. В противном случае следует оставить настройку по умолчанию, поэтому ниже остановимся только на кратком описании вкладок.

Во вкладке **General** производятся сетевые настройки, прописывается путь к директории, в которой находятся базы данных, размер буфера памяти, кодировки и т. д.

Вкладка **Logging** служит для настройки опций сообщений и log-файлов.

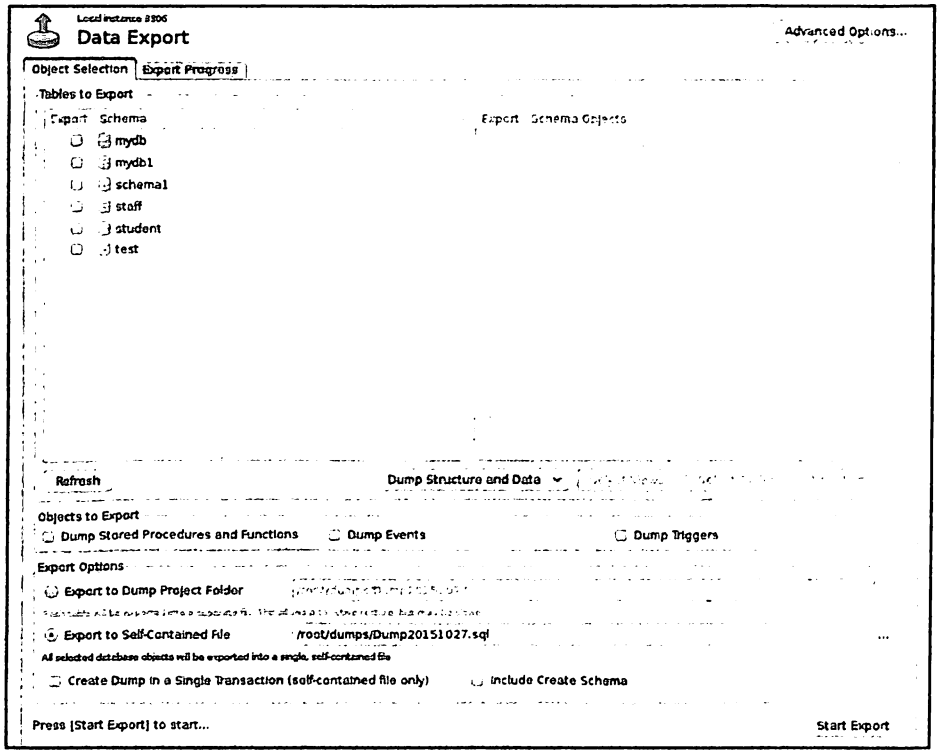


Рис. 3.30. Окно **Data Export**

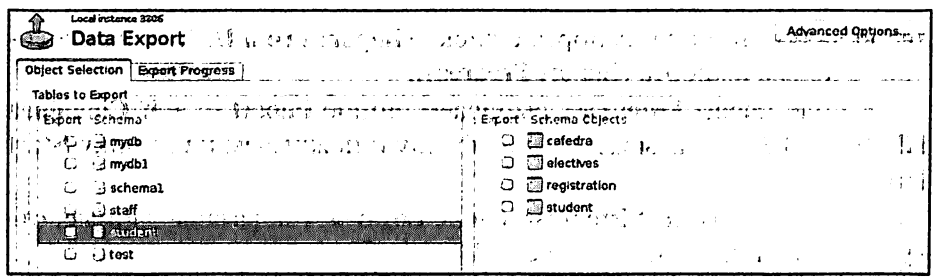


Рис. 3.31. Выбор базы данных для экспорта

Вернемся к вкладке **Instance**. Первый пункт меню **Startup/Shutdown** служит для запуска/остановки сервера. В поле **Startup Message Log** размещена информация о состоянии сервера (рис. 3.41).

При вызове пункта меню **Server Logs** открывается окно с системными сообщениями (рис. 3.42).

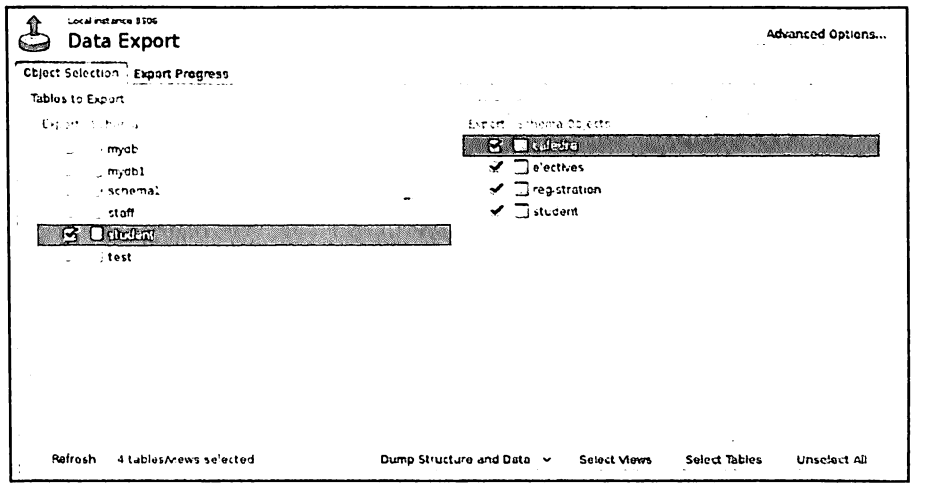


Рис. 3.32. Выбор таблиц для экспорта

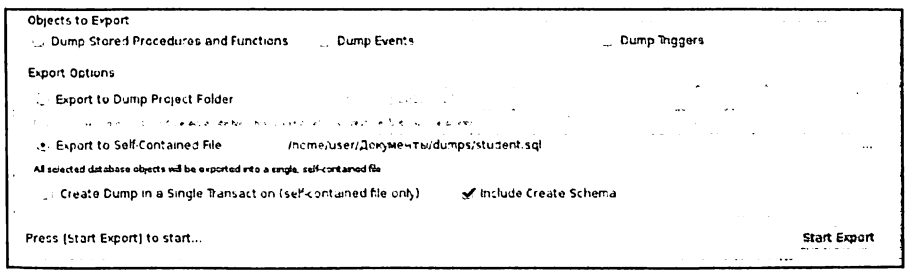


Рис. 3.33. Выбор опций для экспорта в области **Export Options**

Для более тонкой настройки соединения необходимо вызвать пункт меню **Options File** (рис. 3.43).

При выборе пункта меню **Options File** отображается вкладка **General** (общие настройки). Как было сказано ранее, следует производить настройки только в случае, если в этом есть реальная необходимость и пользователь понимает результат своих действий. В противном случае следует оставить настройку по умолчанию, поэтому ниже остановимся только на кратком описании вкладок.

Во вкладке **General** производятся сетевые настройки, прописывается путь к директории, в которой находятся базы данных, размер буфера памяти, кодировки и т. д.

Вкладка **Logging** служит для настройки опций сообщений и **log**-файлов.

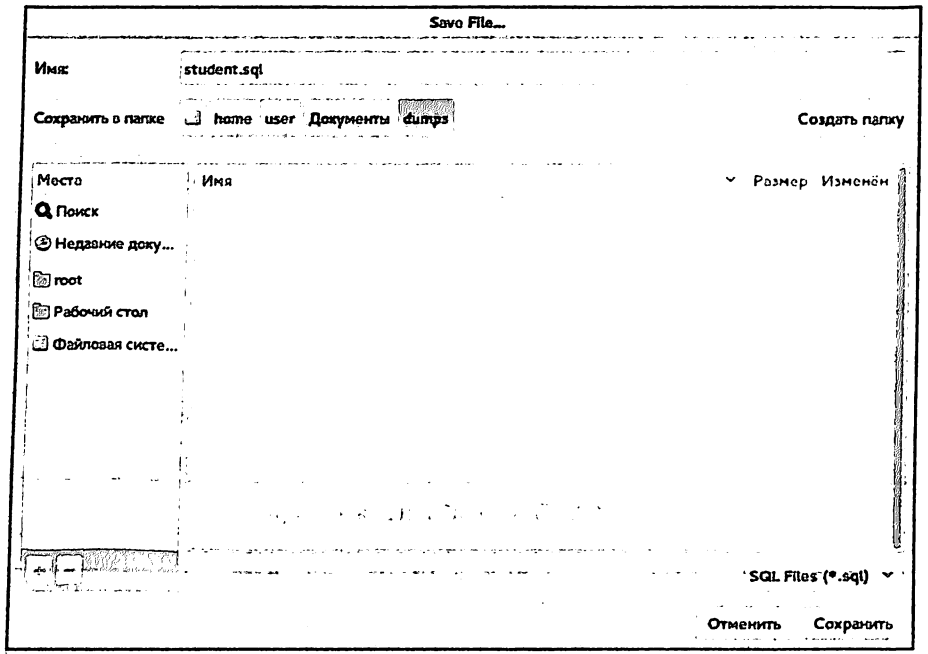


Рис. 3.34. Создание файла для экспорта базы данных

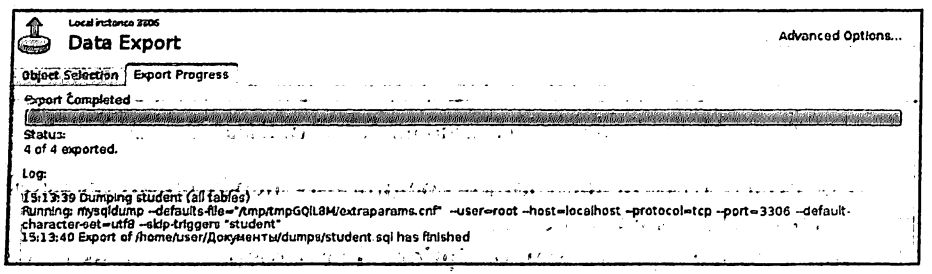


Рис. 3.35. Экспорт данных

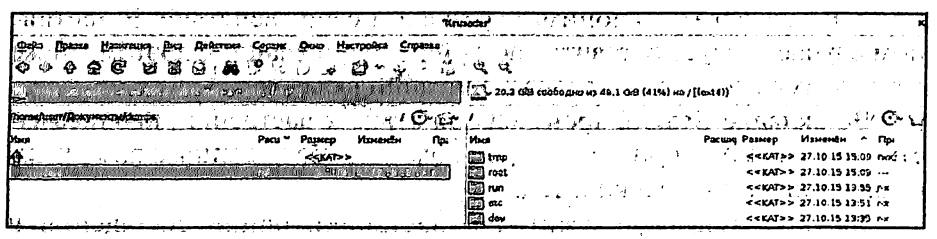


Рис. 3.36. Файл экспортирован

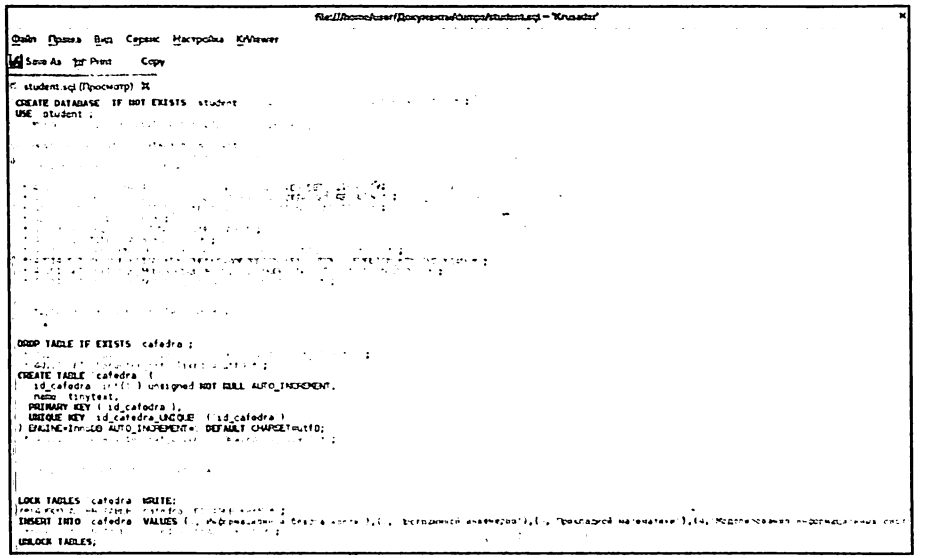


Рис. 3.37. Файл — результат экспорта

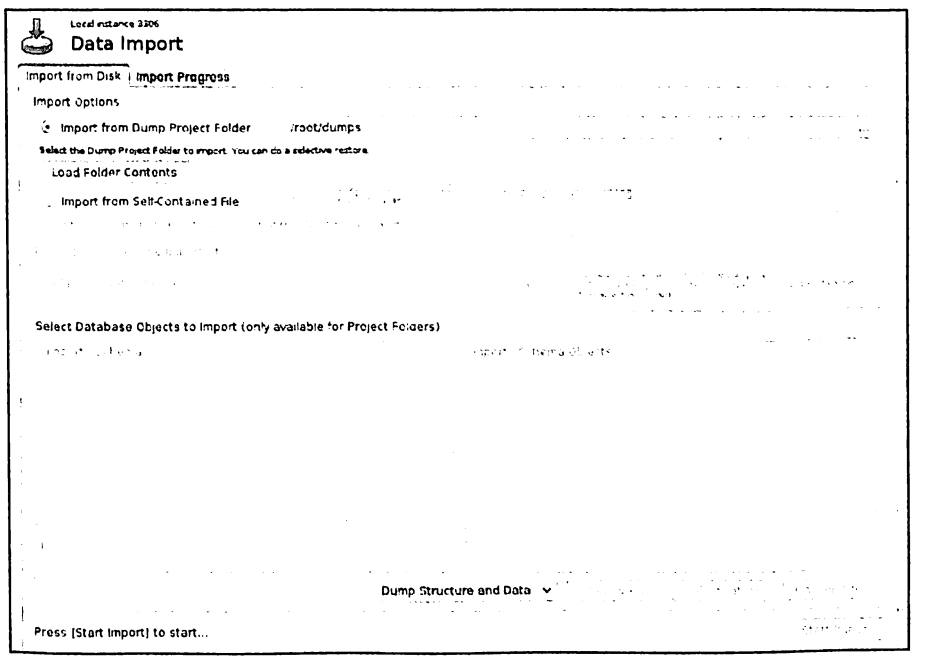


Рис. 3.38. Окно **Import from Disk**

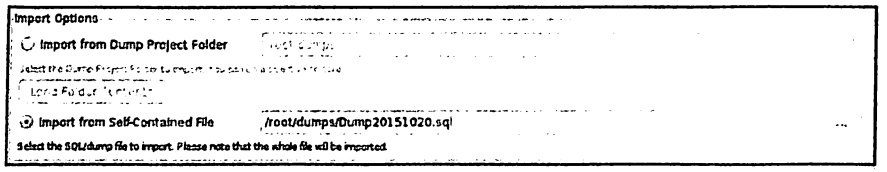


Рис. 3.39. Окно **Import from Disk**

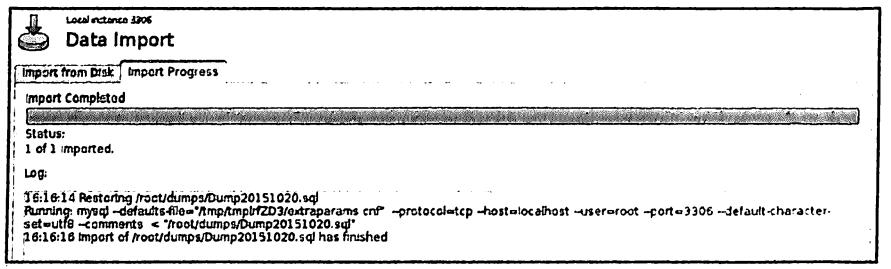


Рис. 3.40. Импорт данных

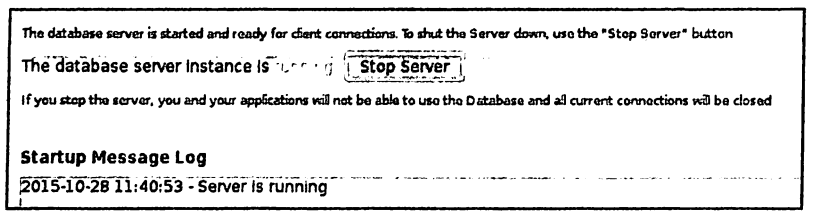


Рис. 3.41. Окно **Startup/Shutdown**

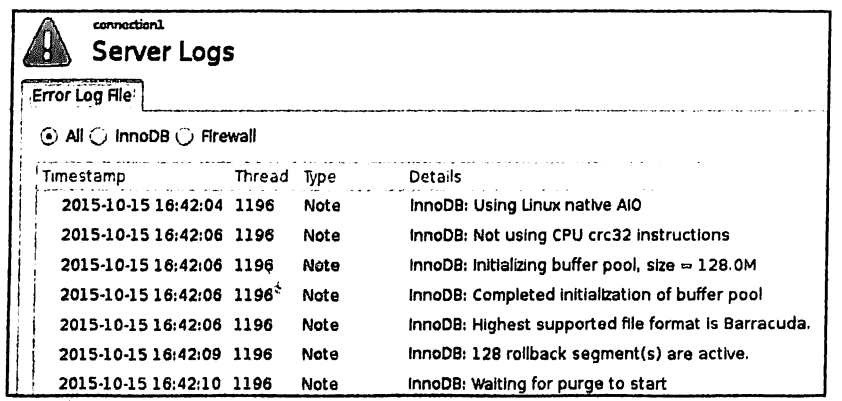


Рис. 3.42. Окно **Server Logs**

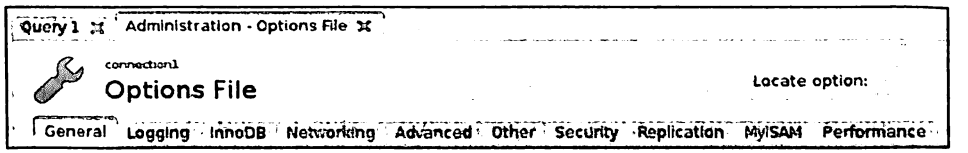


Рис. 3.43. Окно **Options File**

Следующей вкладкой является **InnoDB**. **InnoDB** — это подсистема низкого уровня и входит в состав стандартных сборок основных операционных систем. Отличия при использовании **InnoDB** от **MyISAM** понятны только опытным пользователям, поэтому ограничимся замечанием: традиционно считается, что при использовании многократных изменений/удалений предпочтительнее использовать **InnoDB**, при большом количестве вставок — **MyISAM**. **InnoDB** позволяет произвести настройки, связанные с соединением с базой данных, например автоматическое подтверждение транзакций, использование контрольных сумм для проверки, ограничение числа операций ввода-вывода в секунду и пр.

Для настройки протокола и временных ограничений сетевых соединений служит вкладка **Network**. В ней можно задать максимальный объем данных, передаваемых на сервер, временные ограничения для соединения, ограничения по длительности считывания данных, записи, ожидания, ограничения на число пользователей, одновременно имеющих доступ к серверу, число соединений для конкретного пользователя и т. д.

Далее следует вкладка **Advanced**. В ней производятся дополнительные настройки для транзакций и потоков выполнения.

Other — вкладка, предназначенная для конфигурирования кластера **NDB** — Network Database.

Далее следует вкладка для настроек безопасности **Security**. В ней проводятся настройки аутентификации.

Для репликации данных служит соответствующая вкладка. Репликация (или иначе тиражирование данных) необходима, поскольку в распределенных или клиент-серверных СУБД одни и те же данные хранятся в нескольких узлах. Изменения данных, которые происходят на сервере и повторяются на репликах — копиях базы данных, производятся при помощи бинарных логов, ведущихся на сервере. В этих логах сохраняются все запросы, приводящие к изменениям в БД. При репликации передаются именно эти запросы, а не сами из-

мененные данные. Для настроек свойств репликации и служит вкладка **Replication**.

Вкладка **MyISAM** предназначена для настроек одного из основных способов хранения данных. Второй способ рассмотрен выше. Использование того или иного способа хранения выбирается при настройках соединения и зависит от преобладания основных операций при работе с СУБД. В случае преобладания операций выбора (**SELECT**) предпочтительнее использование **MyISAM**, что связано с отсутствием поддержки транзакций. При большом количестве изменений/удалений предпочтительнее использование **InnoDB**, поскольку при использовании **MyISAM** таблица кратковременно блокируется, что может привести к значительным задержкам при большой загрузке.

Во вкладке **Performance** можно произвести настройки, связанные с таблицами и запросами баз данных.

Таким образом, для успешной работы с СУБД необходимы базовые навыки администрирования сервера MariaDB.

Учащимся предлагается выполнить практическую работу, основанную на изученном теоретическом материале. Необходимо научиться создавать соединение, настроить его, создать нового пользователя с определенными правами доступа, экспортировать и импортировать базу данных. При более углубленном изучении принципов функционирования сервера пользователь сможет более детально разобраться в дополнительных настройках сервера, которые были упомянуты выше.

Порядок выполнения практической работы

1. Ознакомиться с описанием.
2. Запустить **MySQL Workbench** в режиме администрирования.
3. Проверить существующие настройки.
4. Получить у преподавателя задание с именем нового пользователя и перечнем его прав, а также с именем баз данных для выполнения экспорта.
5. Создать нового пользователя с соответствующими правами.
6. Выбрать базу данных для экспорта. Экспортировать таблицу (таблицы) в соответствии с заданием.
7. Получить файл для импорта и импортировать базу данных. Проверить, что база данных импортирована.
8. Сохранить результаты выполнения работы и выйти из программы.

Контрольные вопросы

1. Для чего предназначен режим администрирования?
2. Расскажите, каким образом создается новое соединение. Какие настройки провести необходимо? Какие настройки следует принять по умолчанию?
3. Каким образом можно получить информацию о соединении?
4. Из каких шагов состоит процесс создания нового пользователя?
5. Каким образом назначаются пользователю права доступа?
6. Какие роли могут быть назначены пользователю (назовите не менее трех)?
7. Каким образом осуществляется резервное копирование? Какие настройки необходимо выполнить?
8. Для каких целей используется операция **Data Import**?
9. Какая информация появится во вкладке **Import Progress** в поле **Log** в процессе импорта?
10. Для чего предназначена вкладка **Instance**?

Лабораторная работа 4
СОЗДАНИЕ ER-МОДЕЛЕЙ В MYSQL WORKBENCH

Цель: формирование навыков по работе с инструментом MySQL Workbench для проектирования ER-моделей.

Общие сведения о ER-моделях

При построении баз данных CASE-средства чаще всего ориентированы на два уровня представления модели — логический и физический. **Логическая модель** данных является универсальной и никак не связана с конкретной реализацией СУБД.

Физическая модель данных, напротив, зависит от конкретной СУБД, фактически являясь отображением системного каталога. В физической модели содержится информация обо всех объектах БД. Поскольку стандартов на объекты БД не существует (например, хотя имеется стандарт основных типов данных, в каждой конкретной СУБД могут использоваться дополнительно и свои типы данных), физическая модель зависит от используемой СУБД. Следовательно, одной и той же логической модели могут соответствовать несколько различных физических моделей.

Создание модели данных, как правило, начинается с проектирования **логической модели**. Для проектирования могут использоваться различные инструменты, однако в последнее время наиболее часто применяются CASE-средства (Computer Aided Software Engineering) — средства для автоматизации проектирования программных продуктов.

Как правило, CASE-средства предлагают построение ER-моделей. ER-модель или модель «Сущность—связь» — это аббревиатура от английских слов Entity—Relationship. Приведем определения основных понятий модели, как они даны в [2].

Сущность (Entity) — реальный либо воображаемый объект, имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению.

В ER-модели каждая сущность должна обладать уникальным идентификатором. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от всех других экземпляров данного типа сущности. Каждая сущность должна обладать некоторыми свойствами:

- каждая сущность должна иметь уникальное имя, и к одному и тому же имени должна всегда применяться одна и та же интерпретация. Одна и та же интерпретация не может применяться к различным именам, если только они не являются псевдонимами;
- сущность обладает одним или несколькими атрибутами, которые либо принадлежат сущности, либо наследуются через связь;
- сущность обладает одним или несколькими атрибутами, которые однозначно идентифицируют каждый экземпляр сущности;
- каждая сущность может обладать любым количеством связей с другими сущностями модели.

Атрибут — любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут представляет тип характеристик или свойств, ассоциированных с множеством реальных или абстрактных. Экземпляр атрибута — это определенная характеристика отдельного элемента множества. В ER-модели атрибуты ассоциируются с конкретными сущностями. Таким образом, экземпляр сущности должен обладать единственным определенным значением для ассоциированного атрибута.

Связь (Relationship) — поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области. Связь — это ассоциация между сущностями, при которой, как правило, каждый экземпляр одной сущности, называемой родительской сущностью, ассоциирован с произвольным (в том числе нулевым) количеством экземпляров второй сущности, называемой сущностью-потомком, а каждый экземпляр сущности-потомка ассоциирован в точности с одним экземпляром сущности-родителя. Таким образом, экземпляр сущности-потомка может существовать только при существовании сущности родителя.

Связи может даваться имя, выражаемое грамматическим оборотом глагола и помещаемое возле линии связи. Имя каждой связи между двумя данными сущностями должно быть уникальным, но имена связей в модели не обязаны быть уникальными. Имя связи всегда формируется с точки зрения родителя, так что предложение может быть образовано соединением имени сущности-родителя, имени связи, выражения степени и имени сущности-потомка.

Существуют следующие типы связи: один-к-одному, один-ко-многим и многие-ко-многим. Связь **один-к-одному** отображает такой характер связей между сущностями, когда каждому экземпляру одной сущности соответствует только один экземпляр другой, и наоборот. Наиболее часто встречающимся типом связи в реляционных СУБД является связь **один-ко-многим.** Данный тип связи возникает, когда один экземпляр первой сущности может быть связан с одним или большим количеством экземпляров второй сущности. Связь **многие-ко-многим** описывает случай, когда экземпляры одной сущности связаны с одним или большим количеством экземпляров второй сущности, и наоборот, экземпляры второй сущности связаны с одним или большим количеством экземпляров первой сущности. Связь многие-ко-многим чаще всего используется на начальной стадии проектирования. В дальнейшем при нормализации она чаще всего разбивается при помощи вспомогательных таблиц на несколько связей один-ко-многим.

Для создания ER-диаграмм обычно используют одну из двух наиболее распространенных нотаций:

- **IDEF1X** — усовершенствованная версия IDEF1 (методология структурного анализа для проектирования сложных ИС, разработанная Т. Рэмей (T. Ramey)) и позволяющая разрабатывать концептуальную модель предметной области системы баз данных в форме одной или нескольких **ER-диаграмм**, эквивалентных отношениям в третьей нормальной форме. Помимо того, что эта нотация стала федеральным стандартом США, она также является стандартом в ряде международных организаций (например, Международный валютный фонд и др.) [3, 4];
- **Information Engineering** (IE). Нотация, разработанная Мартином (Martin), Финкельштейном (Finkelstein) и др., используется преимущественно в промышленности.

Заметим, что **IDEF1X** (см. приложение В) является методом для разработки реляционных баз данных и использует нотацию, специально разработанную для удобного построения концептуальной схемы — универсального представления структуры данных в рамках рассматриваемой предметной области, независимого от конечной реализации базы данных и аппаратной платформы. Использование метода IDEF1X наиболее целесообразно для построения логической структуры базы данных.

Основным преимуществом IDEF1X, по сравнению с другими многочисленными методами разработки реляционных баз данных, является жесткая и строгая стандартизация моделирования.

Как было сказано выше, IDEF1X используется для моделирования реляционных баз данных и имеет в США статус федерального стандарта. Стандарт входит в семейство методологий **IDEF** (методологии семейства ICAM (Integrated Computer-Aided Manufacturing) для решения задач моделирования сложных систем), позволяющих исследовать структуру, параметры и характеристики производственно-технических и организационно-экономических систем. Методология IDEF1X адаптирована для совместного использования с IDEF0 (моделирование бизнес-процессов) в рамках единой технологии моделирования. То есть в рамках IDEF0 детализируются функциональные блоки, а в рамках IDEF1X детализируются стрелки, взаимодействующие с функциями.

Сущность описывается в диаграмме IDEF1X графическим объектом в виде прямоугольника. Каждый прямоугольник, отображающий сущность, разделяется горизонтальной линией на часть, в которой расположены ключевые поля, и часть, где расположены остальные атрибуты, не являющиеся ключевыми полями. Верхняя часть называется ключевой областью, а нижняя часть — областью данных. В ключевую область помещают первичный ключ сущности, т. е. атрибуты первичного ключа располагаются над линией в ключевой области. Напомним, что **первичный ключ** — это набор атрибутов, выбранных для идентификации уникальных экземпляров сущности. Остальные атрибуты (неключевые) помещают в области данных (под чертой).

В нотации IDEF1X сущности всегда имеют ключевую область, соответственно, в каждой сущности должны быть определены первичные ключи (ключевые атрибуты).

Определение **первичного ключа** для сущности при проектировании модели данных является очень важным, поскольку помимо идентификации сущности от его выбора зависит удобство дальнейшей работы с данными (вопросы нормализации здесь обсуждаться не будут).

В качестве первичных ключей могут быть использованы один атрибут или группа атрибутов. В начале процесса проектирования, как правило, имеется несколько атрибутов, которые могут быть выбраны первичными ключами. Их называют потенциальными ключами. К потенциальным ключам предъявляется требование уникальной идентификации каждого экземпляра сущности. Соответственно, ни одна из частей ключа не может принимать неопределенное значение (NULL).

В некоторых случаях при разработке модели удобно использовать так называемый суррогатный ключ, т. е. вводят искусственно некоторый атрибут, который уникальным образом определяет экземпляр сущности. Примером такого искусственно введенного ключа может являться атрибут «Номер по порядку» или «Номер записи». Именно такой ключ лучше всего подходит на роль первичного ключа, потому что является целочисленным, коротким и по нему можно быстро производить поиск и идентификацию экземпляров в сущности. Более того, практически все современные СУБД имеют в своем арсенале возможности для автоматической генерации такого рода ключей. Это особенно удобно в тех случаях, когда необходима сплошная нумерация.

При изучении предметной области и выборе первичных ключей некоторые атрибуты могут стать потенциальными ключами, т. е. атрибуты (или совокупность атрибутов), которые не выбраны первичными ключами. Такие ключи называют альтернативными ключами. Однако следует иметь в виду, что, хотя все потенциальные ключи могут быть использованы в качестве первичного ключа, желательно в качестве первичного ключа выбирать те потенциальные ключи, которые требуют меньше памяти для хранения и состоят из меньшего числа атрибутов.

Сущность в методологии IDEF1X является независимой от идентификаторов или просто **независимой**, если каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями.

В нотации IDEF1X независимые сущности представлены на диаграмме в виде прямоугольников.

При связывании сущностей в нотации IDEF1X родительская сущность передает ключ (или набор ключевых атрибутов) дочерней сущности. Такие ключи — атрибуты дочерней сущности называются внешними ключами, и в нотации IDEF1X на диаграмме рядом с внешними ключами возникает пометка (**FK**), что означает **Foreign Key** — внешний ключ.

Для создания связи между независимыми сущностями служат неидентифицирующие связи (см. выше).

Неидентифицирующие связи отображаются в виде пунктирной линии между сущностями (прямоугольниками). Они связывают родительскую и дочернюю сущности, но при этом не являются составной частью первичного ключа. В этом случае внешний ключ отображается в области данных под линией.

В некоторых случаях при анализе предметной области выделяются сущности, уникальность которых зависит от значений атрибута некоторой другой сущности. Для уникального определения таких сущностей необходим внешний ключ, который является ключом родительской сущности и должен быть частью первичного ключа дочерней сущности. Дочерняя сущность, уникальность которой зависит от атрибута внешнего ключа, называется **зависимой сущностью**.

В нотации IDEF1X зависимые сущности представлены на диаграмме в виде прямоугольников со скругленными углами.

Если необходимо при помощи внешнего ключа идентифицировать дочернюю сущность, то создается идентифицирующая связь между родительской и дочерней сущностью.

Идентифицирующие взаимосвязи обозначаются сплошной линией между сущностями. В этом случае внешний ключ отображается в области первичного ключа над линией. Поскольку внешний ключ не может принимать значение NULL, запись в дочерней сущности может существовать только при наличии ассоциированной с ним родительской записи.

Для связи можно также определить мощность — т. е. количество экземпляров сущности-потомка, которое может существовать для каждого экземпляра сущности-родителя. В IDEF1X могут быть отображены следующие мощности связей.

1. Каждый экземпляр сущности-родителя может иметь ноль, один или более связанных с ним экземпляров сущности-потомка.
2. Каждый экземпляр сущности-родителя должен иметь не менее одного связанного с ним экземпляра сущности-потомка.
3. Каждый экземпляр сущности-родителя должен иметь не более одного связанного с ним экземпляра сущности-потомка.
4. Каждый экземпляр сущности-родителя связан с некоторым фиксированным числом экземпляров сущности-потомка (обозначается N).

Рабочее пространство для проектирования ER-моделей в MySQL Workbench

Рассмотрим основные принципы работы с MySQL Workbench для построения ER-моделей. При работе с ER-моделями Workbench позволяет:

- создавать графическую модель базы данных;
- при помощи прямого инжиниринга создавать из графической модели базу данных;
- при помощи обратного инжиниринга из существующей базы данных получать графическую модель.

Вход в MySQL Workbench осуществляется нажатием соответствующей иконки на рабочем столе или через меню «Пуск». Далее для работы с ER-моделями необходимо выбрать для входа область моделирования данных, расположенную в нижнем правом углу (**Models**), и нажать на кнопку ⊕ для создания новой модели (рис. 4.1).

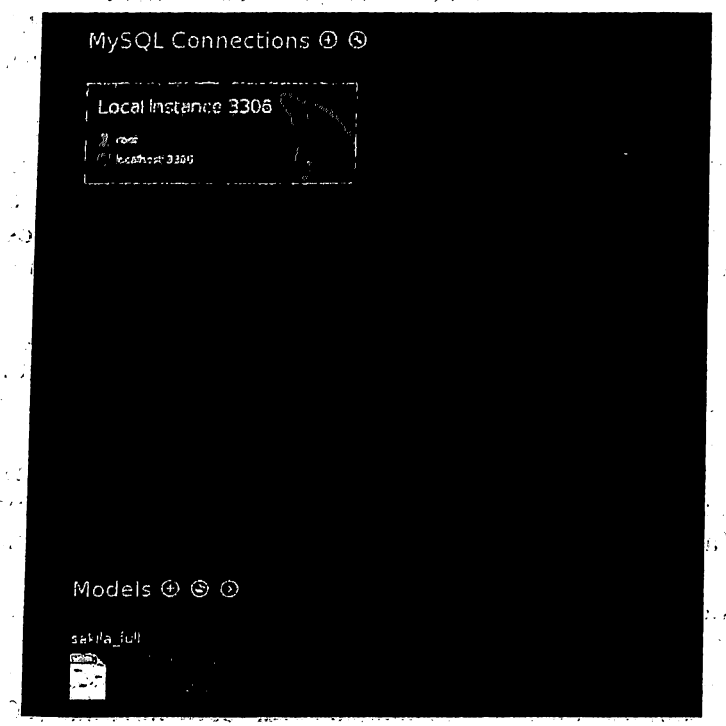


Рис. 4.1. Создание новой ER-модели

После запуска будет открыто окно редактирования модели (рис. 4.2), которое будет рассмотрено ниже.

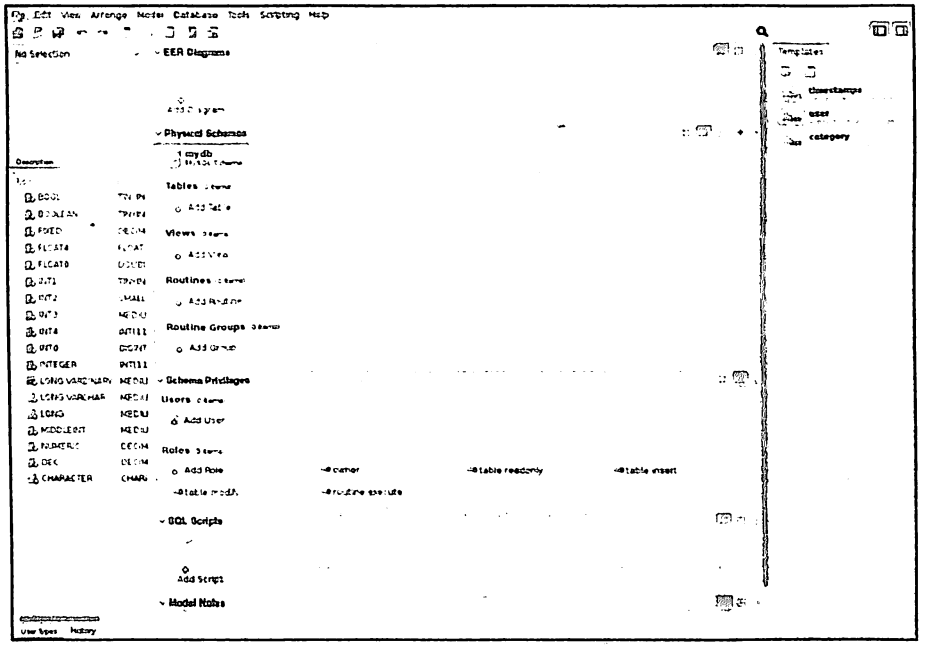


Рис. 4.2. Окно редактирования ER-модели

Если нажать на кнопку ⊖, то откроется окно для выбора файла, в котором хранится созданная ранее модель (рис. 4.3).

Если нажать на кнопку ⊙, то можно выбрать либо создание ER-модели из существующей базы данных (т. е. обратный инжиниринг), либо создание ER-модели из существующего скрипта (рис. 4.4).

Обратите внимание, что в данном случае построение модели ведется в EER (enhanced entity-relationship) редакторе. Основное отличие EER-моделирования от ER-моделирования состоит в том, что EER поддерживает понятия подкласса и суперкласса, а также связанных с ними понятий специализации и обобщения. Однако в большинстве случаев для проектирования баз данных эти возможности являются излишними.

Рассмотрим подробнее окно редактирования ER-модели (рис. 4.2). В окне присутствуют четыре основные подобласти: слева **Description Edior** (редактор описаний) и **Users Types/History** (окно для пользова-

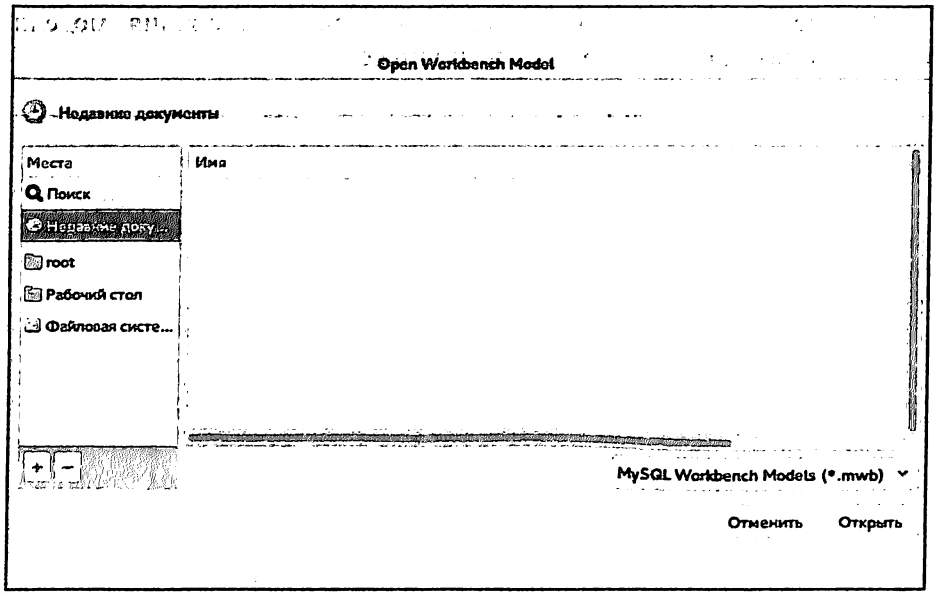


Рис. 4.3. Окно выбора файла

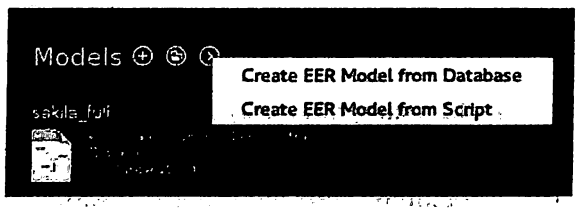


Рис. 4.4. Переход к созданию модели из существующей базы данных или скрипта

тельских типов данных/истории модели), справа **Templates** (шаблоны) и основное окно.

В этом основном окне присутствуют вкладки:

- **ERR Diagram** (для добавления диаграммы);
- **Physical Schemata** (физическая схема);
- **Schema Privileges** (привилегии);
- **SQL Scripts** (SQL-скрипты);
- **Model Notes** (заметки по модели).

В каждой из этих секций можно выбирать объекты для добавления их в модель. Например, в секции **Physical Schemata** можно добавлять таблицы, вид, программу, группу (рис. 4.5).

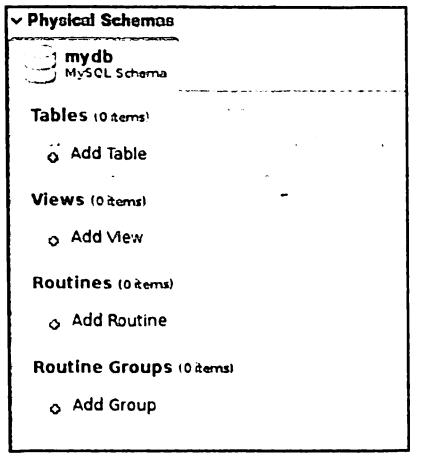


Рис. 4.5. Окно редактирования модели, секция **Physical Schemata**

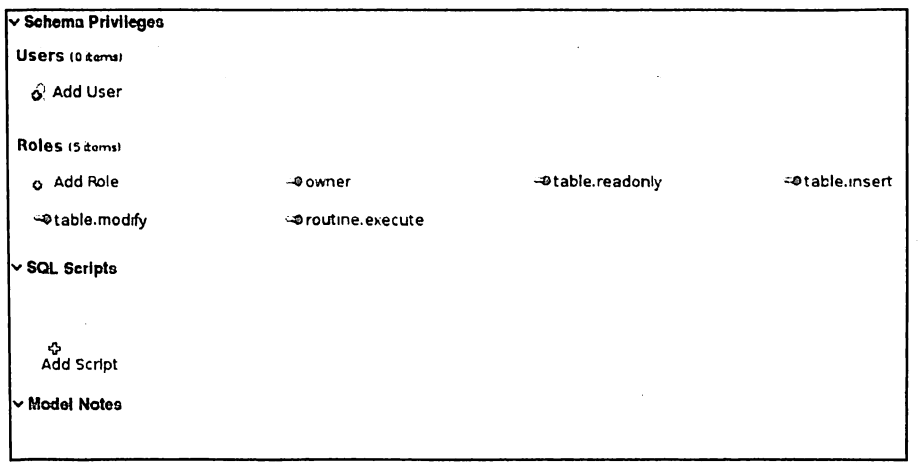


Рис. 4.6. Окно редактирования модели, секции **Schema Privileges**, **SQL Scripts** и **Model Notes**

В секции задания привилегий можно задавать пользователей и роли, в секции **SQL scripts** — скрипты, в секции **Model Notes** — замечания по модели (рис. 4.6).

В верхней левой части вкладок присутствуют кнопки, позволяющие настроить размер иконок (большие или маленькие, рис. 4.7), расположение объектов в виде списка (рис. 4.8).

Рис. 4.7. Настройка отображения объектов во вкладке

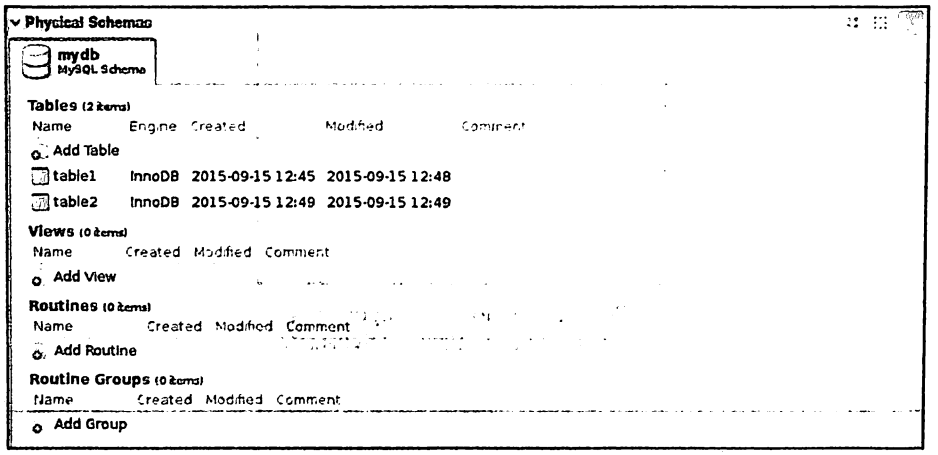


Рис. 4.8. Настройка отображения объектов списком

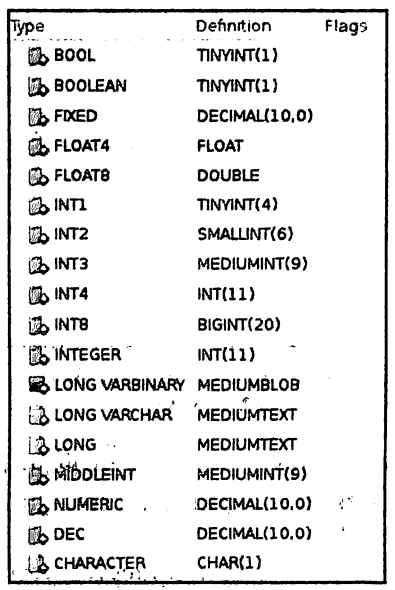


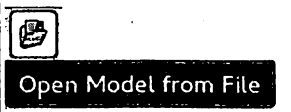
Рис. 4.9. Окно редактирования модели, подобласть **Users Types/History**

Слева в подобласти **Users Types/History** (рис. 4.9) показаны типы данных пользователей.

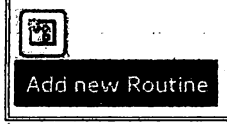
Опции основного меню являются аналогичными опциям меню для ER-модели и будут рассмотрены далее.

Ниже под пунктами основного меню расположены пиктограммы (табл. 4.1), позволяющие выполнять некоторые действия непосредственно по щелчку мыши, а не через пункты меню. Некоторые пункты меню имеют уже известное назначение.

Таблица 4.1. **Элементы управления панели инструментов**

Пиктограммы и горячие клавиши элементов управления	Описание
	Создать новый документ (модель)
	Загрузить модель из файла
	Сохранить модель в текущем файле
	Отмена действия
	Возврат отмененного действия
	Добавить новую диаграмму
	Добавить новую схему

Окончание табл. 4.1

Пиктограммы и горячие клавиши элементов управления	Описание
	Добавить новую таблицу
	Добавить новый вид
	Добавить новую программу

Если разработчик выбрал добавление ER-модели, то перед ним открывается основное окно программы для построения ER-моделей (рис. 4.10).

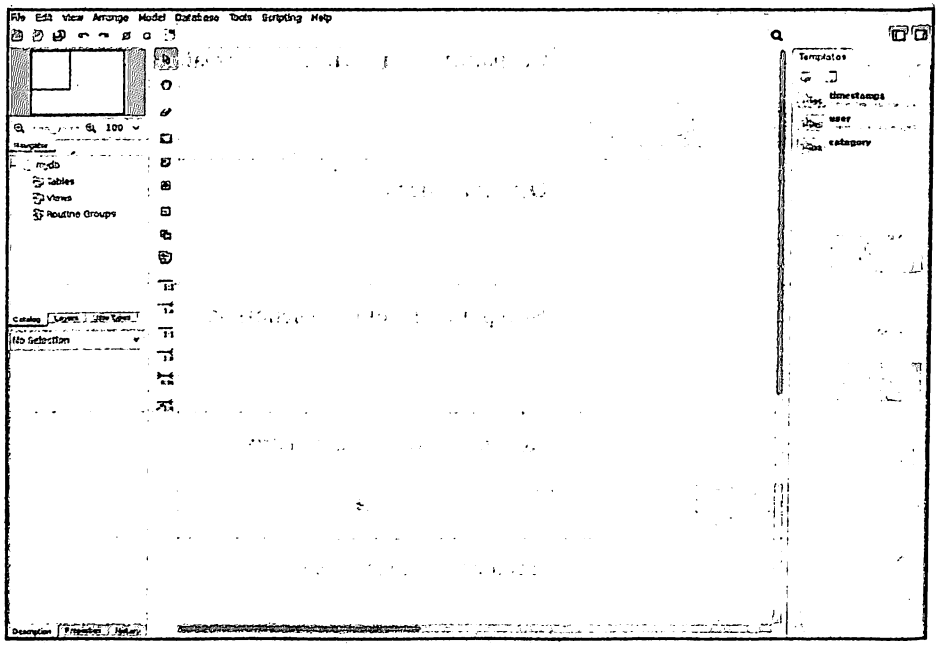


Рис. 4.10. Окно **EER Diagram**

В верхней части окна расположено меню, которое является достаточно традиционным, тем не менее рассмотрим его подробнее.

Пункт меню работы с файлом (**File**) позволяет (рис. 4.11) создавать новую модель, создавать новую модель из скриптов, открывать существующую модель, открывать модель, с которой недавно проходила работа, сохранять модель и сохранять модель под другим именем, экспортировать модель в файл следующего формата: PNG, SVG, PDF или PostScript для последующей печати (рис. 4.12), распечатывать модель, задавая параметры печати, а также импортировать скриптовые файлы DDL для обратного инжиниринга (рис. 4.13).

Далее следует пункт меню Редактирование (**Edit**). С помощью пунктов подменю можно отменять действия (горячие клавиши хорошо известны: CTRL+Z и CTRL+Y), копировать, вырезать, вставлять, удалять, редактировать выделенные фрагменты, если выделенных фрагментов несколько, то перемещаться между ними, осуществлять поиск, настраивать предпочтения (рис. 4.14). Если для данного объекта некоторое действие выполнено быть не может, то подпункт

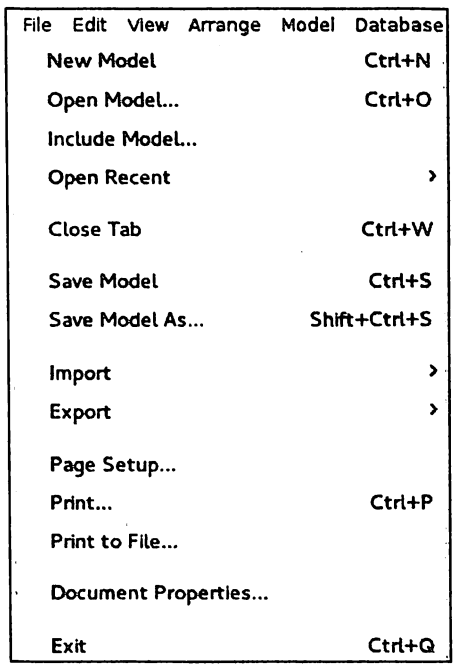


Рис. 4.11. Пункт меню **File**

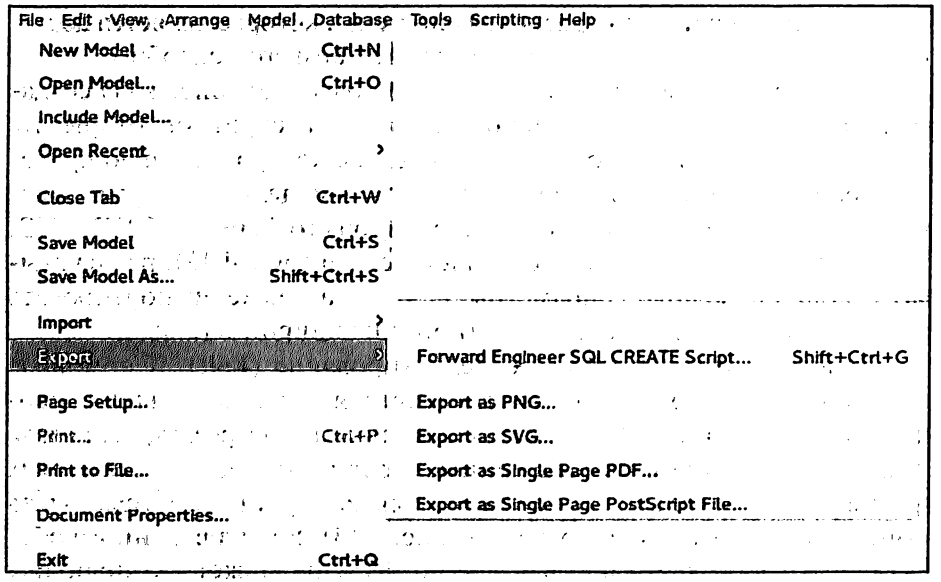


Рис. 4.12. Пункт меню **File, Export**

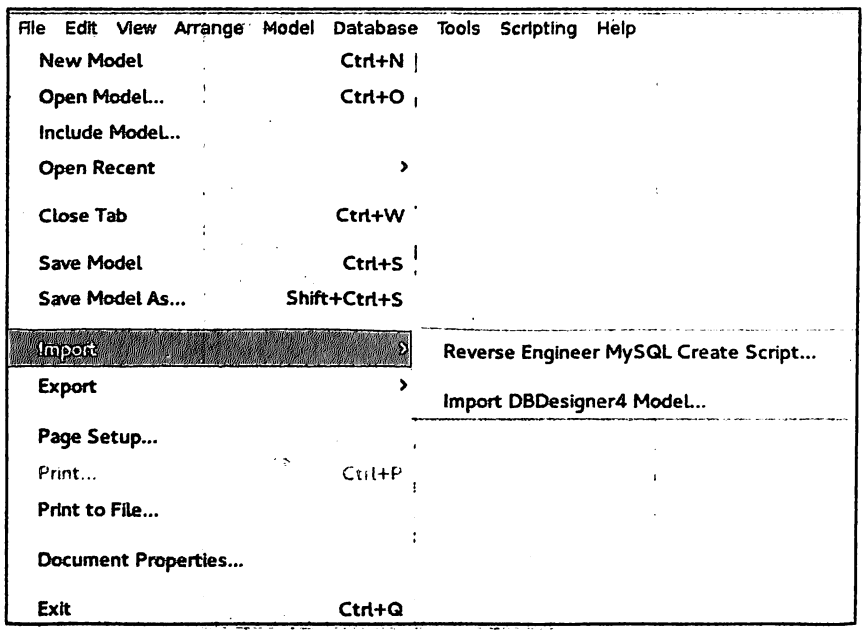


Рис. 4.13. Пункт меню **File, Import**

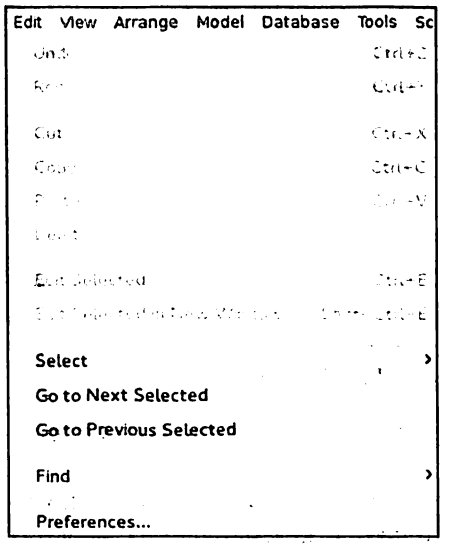


Рис. 4.14. Пункт меню **Edit**

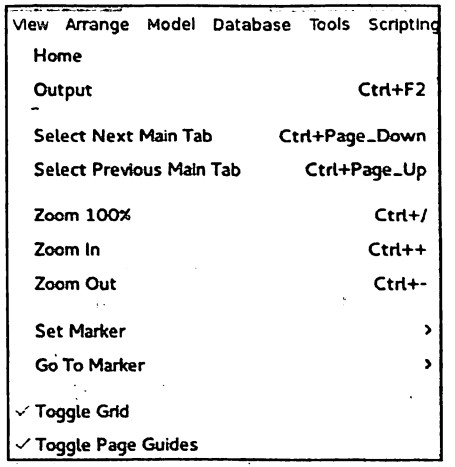


Рис. 4.15. Пункт меню **View**

меню будет неактивным. Пункт **Preferences**, как было сказано ранее, позволяет выполнить пользовательские настройки редактора.

Для настроек отображения на экране служит пункт меню Вид — **View** (рис. 4.15). При помощи данного пункта меню можно переходить на экран Home, увеличивать и уменьшать изображение, включать разметку сетки, выравнивать изображение по сетке, перемещаться между таблицами.

Для настройки расположения изображения, его растяжения и сжатия служит пункт меню Размещение — **Arrange** (рис. 4.16). Данный пункт меню применим только к активным (выделенным) объектам на диаграмме.

Далее следует пункт меню Модель — **Model**, который позволяет произвести все необходимые настройки для работы с моделями: добавить новую диаграмму, создать диаграмму из каталога объектов, использовать данные, определенные пользователем, использовать шаблоны таблиц, выбрать нотацию для объектов и связей, настроить размер и свойства диаграмм, установить опции для уровня модели, которые, однако, не должны вступать в противоречие с общими настройками модели (рис. 4.17).

Пункт меню База данных — **Database** позволяет переходить на вкладку для составления запросов, управлять соединением, осущест-

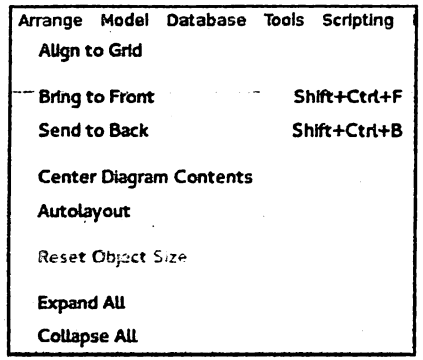


Рис. 4.16. Пункт меню **Arrange**

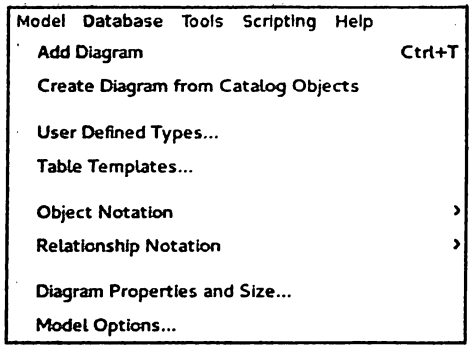


Рис. 4.17. Пункт меню **Model**

влять прямой и обратный инжиниринг, синхронизировать модели, сравнивать схемы (рис. 4.18).

Далее следует пункт меню Инструменты — **Tools**, функции которого в полном объеме доступны только в коммерческих версиях MySQL (рис. 4.19).

Для работы со скриптами предназначен пункт меню Скрипты — **Scripting** (рис. 4.20). Этот пункт меню позволяет запускать скриптовую оболочку, создавать новый скрипт, открывать имеющийся, выполнять определенный скриптовый файл, загружать и выполнять плагин/модуль, управлять работой с плагинами.

Пункт меню Помощь — **Help** (актуален в том случае, если имеется доступ к сети Интернет) содержит справочную информацию по про-

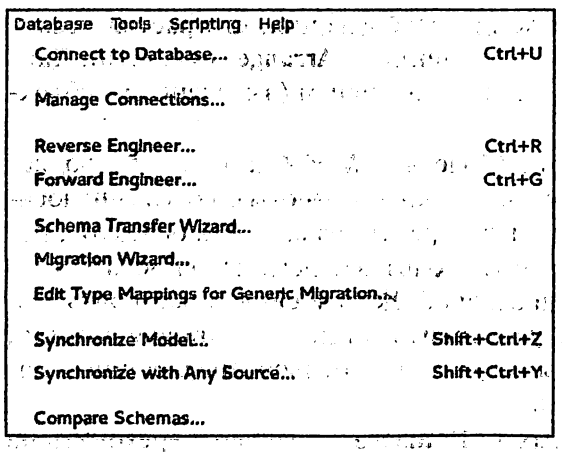


Рис. 4.18. Пункт меню База данных — **Database**

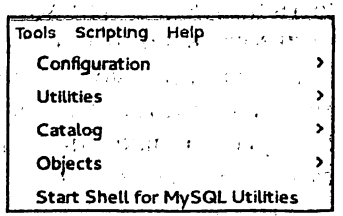


Рис. 4.19. Пункт меню **Tools**

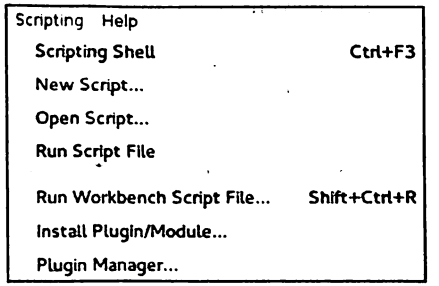


Рис. 4.20. Пункт меню Скрипты — **Scripting**

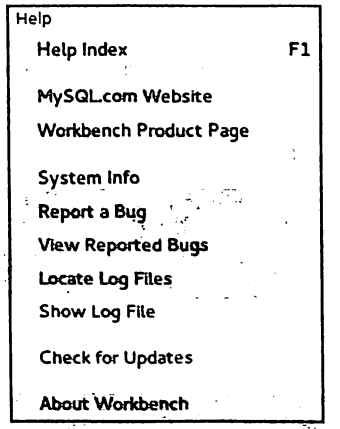


Рис. 4.21. Пункт меню **Help**

дукту (рис. 4.21). Вызов справки традиционно осуществляется при помощи клавиши **F1**.

Далее рассмотрим вертикальную панель управления, служащую для создания и редактирования ER-диаграмм. Элементы управления приведены в табл. 4.2. В табл. 4.3 приведены связи, пиктограммы которых (и горячие клавиши) расположены на вертикальной панели инструментов под элементами управления, приведенными в табл. 4.2.

Таблица 4.2. **Вертикальная панель управления**

Пиктограммы и горячие клавиши элементов управления	Описание
Select Object(s) (Quick key - press Escape)	Стандартный указатель мыши, на него следует переключаться для снятия любого выделения иного пункта меню
Move Model (Quick key - press H)	Рука служит для перемещения ER-модели по диаграмме
Delete Object (Quick key - press D)	Ластик служит для удаления объектов с диаграммы

Окончание табл. 4.2

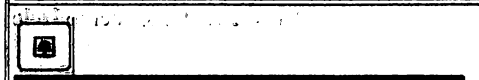
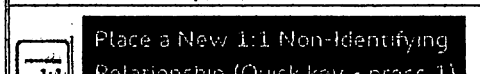
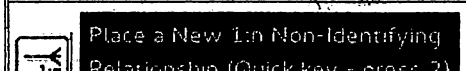
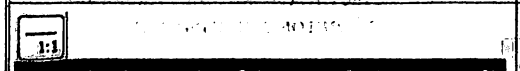
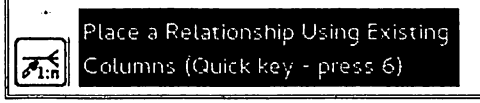
Пиктограммы и горячие клавиши элементов управления	Описание
	Инструмент создания слоев служит для создания объектов на диаграмме. После того как объект поместили на диаграмме, можно изменять размеры, растягивая его. Также можно произвести настройки изображения
	Данная пиктограмма служит для создания текста на диаграмме
	Данная пиктограмма служит для помещения рисунков на диаграмму
	Для создания новой таблицы на диаграмме используется следующая пиктограмма
	Для настройки изображения следует использовать пиктограмму View
	Помещение новой области программ

Таблица 4.3. **Вертикальная панель управления. Связи**

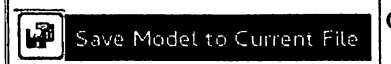
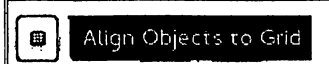
Пиктограммы и горячие клавиши элементов управления	Описание
	Создание неидентифицирующей связи один-к-одному
	Создание неидентифицирующей связи один-ко-многим
	Создание идентифицирующей связи один-к-одному

Окончание табл. 4.3

Пиктограммы и горячие клавиши элементов управления	Описание
	Создание идентифицирующей связи один-ко-многим
	Создание идентифицирующей связи многие-ко-многим
	Создать связь, используя имеющиеся столбцы

И наконец, — элементы управления, которые расположены под главным меню. Они приведены в табл. 4.4. Некоторые пиктограммы имеют уже известное назначение.

Таблица 4.4. **Элементы управления панели инструментов**

Пиктограммы и горячие клавиши элементов управления	Описание
	Создать новый документ (модель)
	Загрузить модель из файла
	Сохранить модель в текущем файле
	Отмена действия
	Возврат отмененного действия
	Включить отображение сетки
	Выровнять объекты по сетке
	Добавить новую диаграмму

Построение тестовой ER-модели

Построим тестовую ER-модель, состоящую из двух связанных между собой таблиц. Для этого сначала поместим на диаграмму таблицу **table1** (рис. 4.22), щелкнув мышью на соответствующем значке пиктограммы, а затем на рабочем пространстве окна. Можно использовать горячую клавишу T.

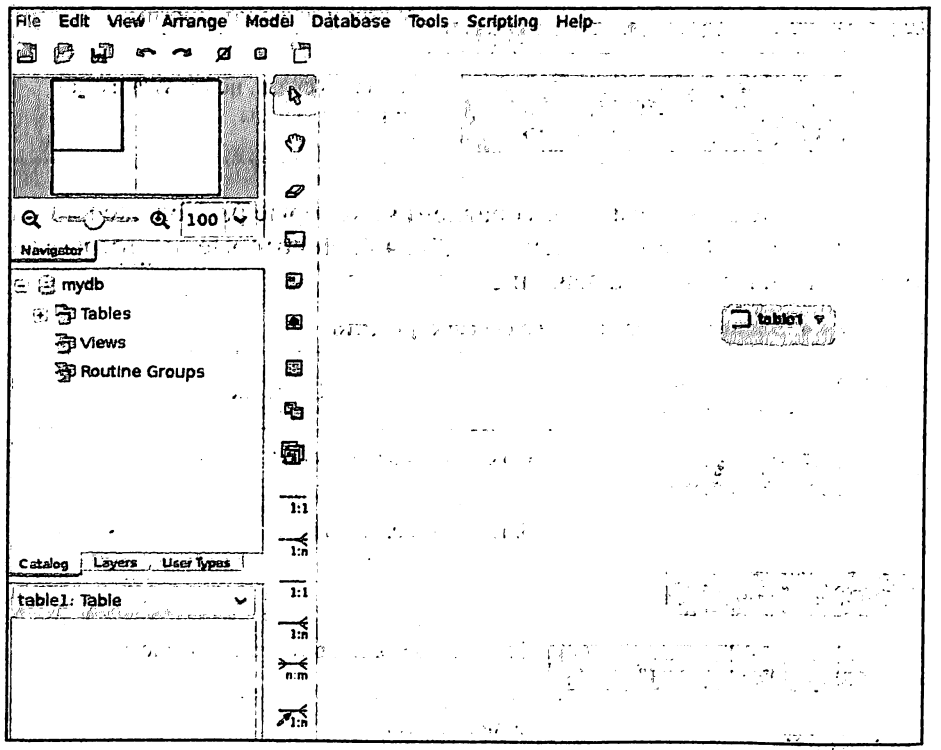


Рис. 4.22. Создание таблицы в рабочем пространстве **EER Diagram**

Далее выделим таблицу 1 (щелкнем на ней мышью, чтобы сделать объект активным), после этого внизу экрана откроется окно **Table table1** для работы с таблицей (рис. 4.23). В нем можно создавать столбцы, индексы, внешние ключи и т. п.

Перейдем на вкладку **Columns** окна **Table table1** для создания столбцов таблицы — полей базы данных. Для простоты первичный ключ назовем **idtable1** (тип данных — целое, автоинкремент), а столбец **table1col** (тип данных — строковый) (рис. 4.24). У столбцов имеет-

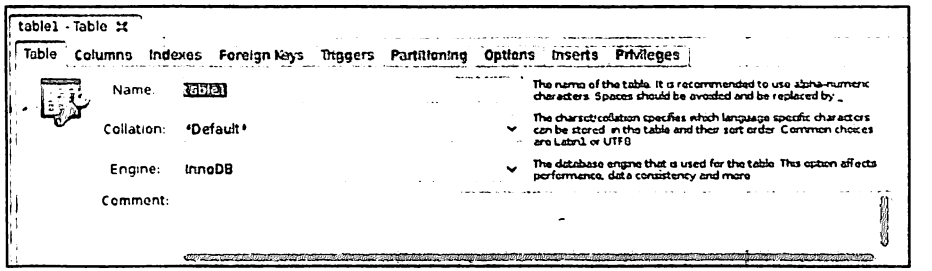


Рис. 4.23. Окно **Table table1** для работы с таблицей

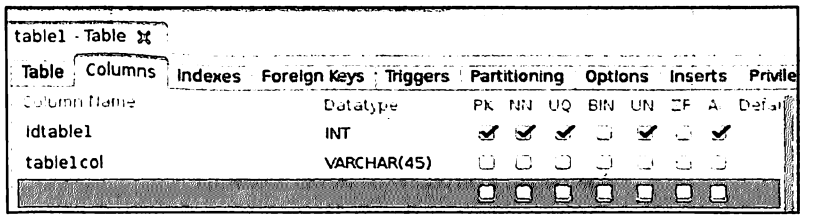


Рис. 4.24. Задание столбцов таблицы

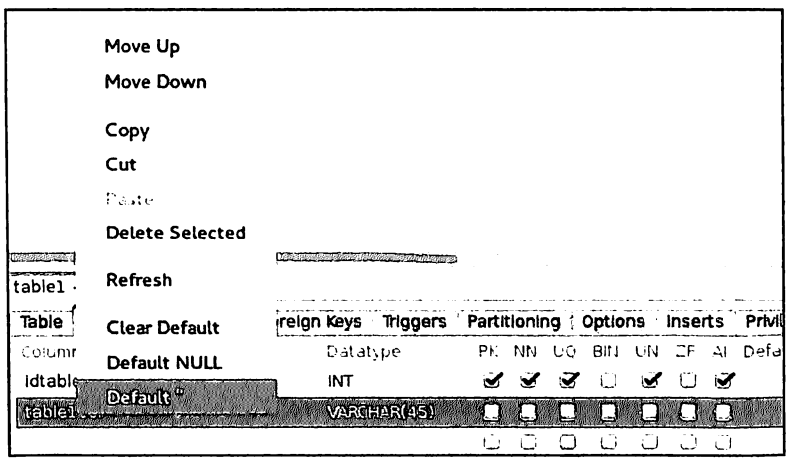


Рис. 4.25. Контекстное меню для столбцов таблицы

ся контекстное меню, которое служит для их редактирования и перемещения (рис. 4.25).

Заметим, что для таблицы, как и любого другого объекта, имеется возможность вызова контекстного меню при помощи правой кнопки мыши (рис. 4.26).

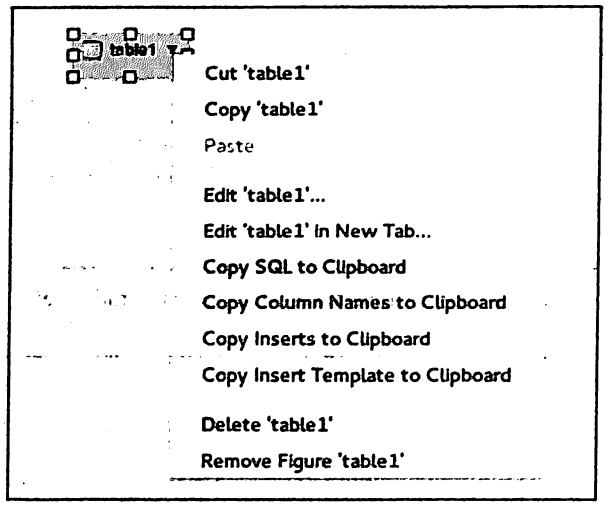


Рис. 4.26. Контекстное меню таблицы

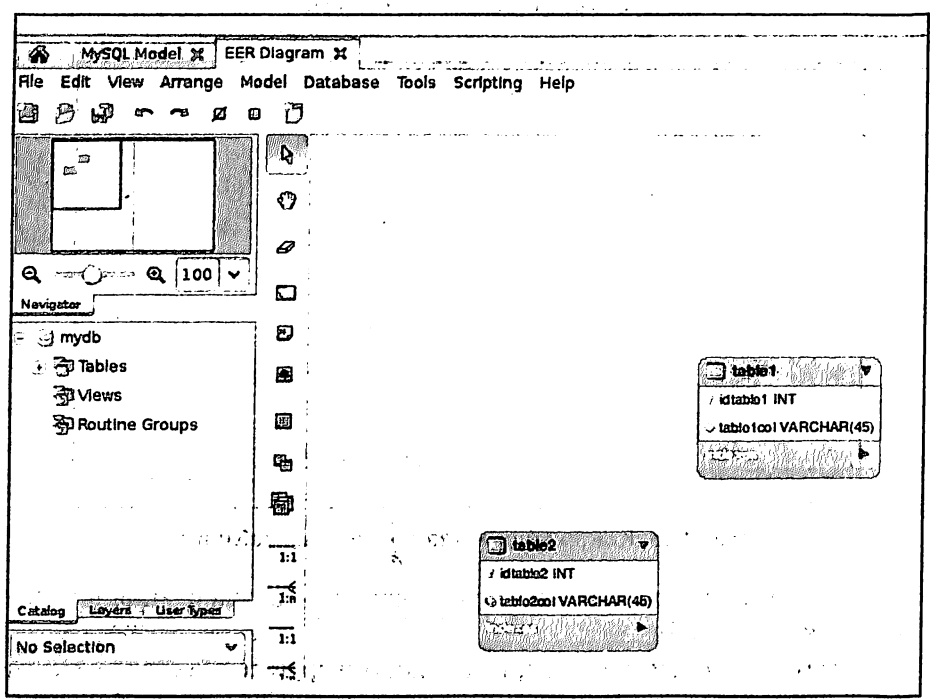


Рис. 4.27. Созданные таблицы

Аналогично создадим таблицу **table2** и атрибуты для нее. После чего в окне редактора ER-диаграмм появятся обе таблицы с созданными первичными ключами, помеченными значком , и остальными атрибутами (рис. 4.27).

Далее произведем настройку нотаций. Выберем нотацию IDEF1X для отображения таблиц. Для этого вызовем в меню пункт **Model → Object Notation → IDEF1X** (рис. 4.28).

Также выберем нотацию IDEF1X для отображения связей. Для этого вызовем в меню пункт **Model → Relationship Notation → IDEF1X** (рис. 4.29).

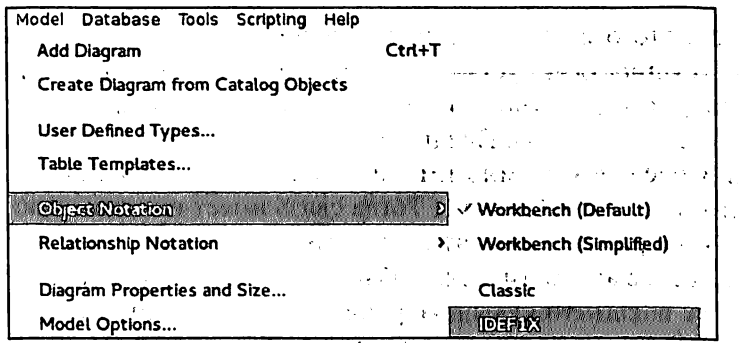


Рис. 4.28. Выбор нотации **IDEF1X** для отображения таблиц

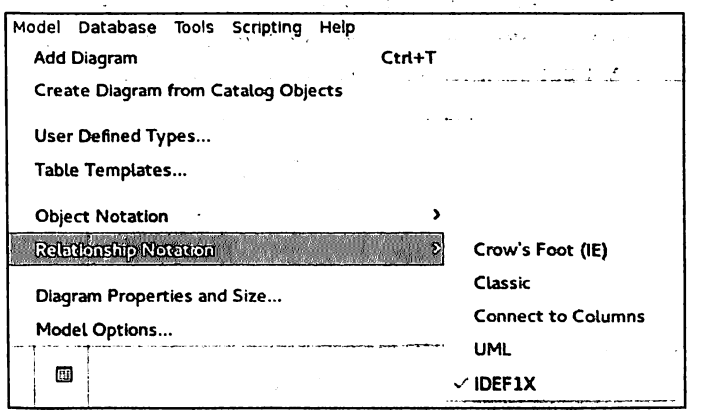


Рис. 4.29. Выбор нотации **IDEF1X** для отображения связей

В результате таблицы и построенные далее связи будут отображены в нотации **IDEF1X** (рис. 4.30).

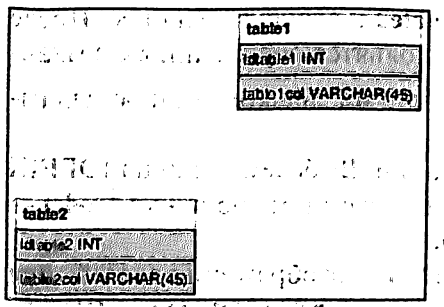


Рис. 4.30. Отображение таблиц в нотации **IDEF1X**

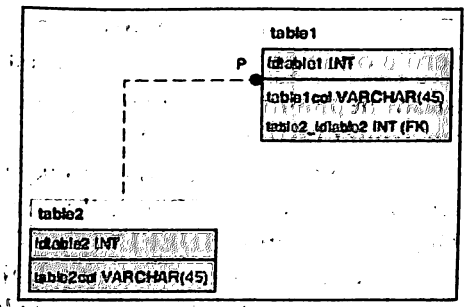


Рис. 4.31. Связь между таблицами

После этого можно соединить таблицы. Очевидно, что между таблицами (сущностями) устанавливаются неидентифицирующие связи, поскольку каждая сущность имеет свой уникальный идентификатор — первичный ключ. Предположим, что связь между данными таблицами у нас один-ко-многим и первичный ключ таблицы 2 является внешним ключом таблицы 1. Тогда связь имеет вид, как показано на рис. 4.31. Обратите внимание, что при построении первичный ключ таблицы **table2** автоматически добавляется к атрибутам таблицы **table1** в качестве внешнего ключа. Первичный ключ в соответствии с нотацией **IDEF1X** расположен в верхней части прямоугольника, отображающего таблицу, а внешний ключ — в нижней вместе с остальными атрибутами и помечен аббревиатурой FK (Foreign Key).

Контекстное меню, как и в предыдущих случаях, может быть вызвано щелчком правой кнопкой мыши на связи (рис. 4.32).

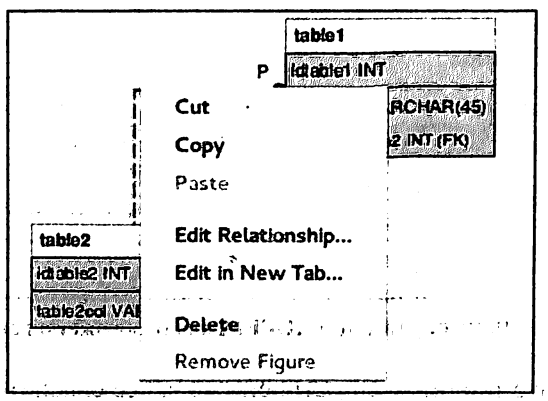


Рис. 4.32. Контекстное меню связи

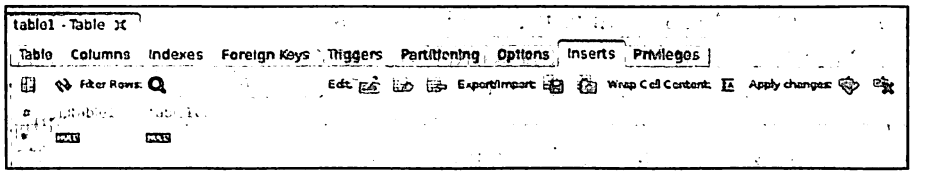


Рис. 4.33. Вкладка **Inserts** для заполнения таблицы

Построение тестовой ER-модели завершено. Для заполнения таблицы можно перейти на соответствующую вкладку **Inserts** (рис. 4.33).

Далеё рассмотрим случай построения идентифицирующих связей. В этом случае третья таблица будет идентифицироваться при помощи внешнего ключа. Для этого создадим три таблицы **table1**, **table2** и **table3**. У первых двух таблиц создадим такие же первичные ключи **idtable1** и **idtable2** и атрибуты **table1col** и **table2col**. Пусть у третьей таблицы первичный ключ будет составным **idtable1** и **idtable2**, что обеспечит его уникальность. А атрибут — **table3col**. Для создания первичного ключа поместим на диаграмму с незаполненным ключевым полем таблицу **table3** и построим идентифицирующие связи (рис. 4.34). Обратите внимание, что при построении связей автоматически получим составной первичный ключ, который будет состоять из двух внешних ключей таблиц **table1** и **table2**. Оба внешних ключа, составляющие уникальный первичный ключ таблицы **table3**, в данном случае в соответствии с нотацией располагаются в поле первичного ключа, а атрибут — в поле атрибутов. Также обратите внимание, что в соответствии

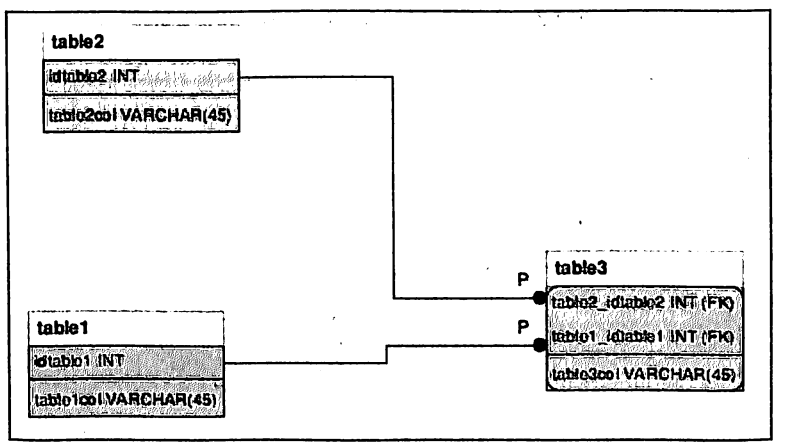


Рис. 4.34. Идентифицирующие связи

с нотацией **IDEF1X** зависимая таблица **table3** будет представлена в виде прямоугольника со скругленными углами.

Если посмотреть на вкладку **Columns** таблицы **table3**, то увидим, что при построении связей поля первичного ключа были созданы (рис. 4.35). И появилась информация во вкладке **Foreign Key** (рис. 4.36).

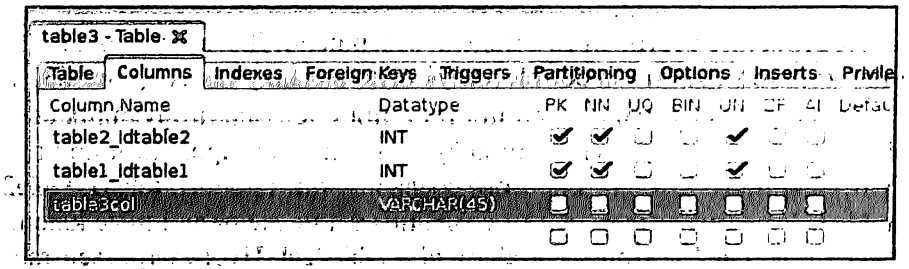


Рис. 4.35. Вкладка Columns таблицы table3

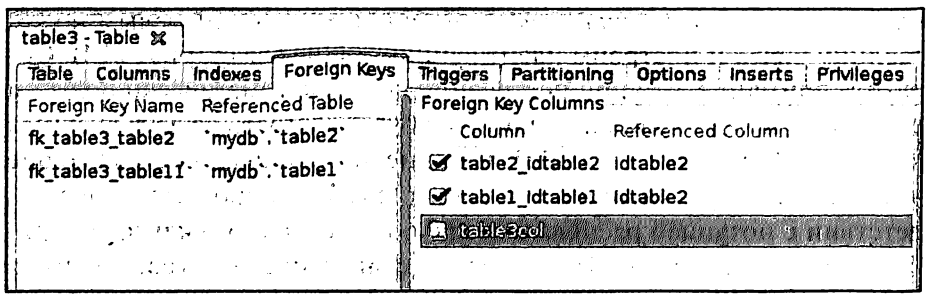


Рис. 4.36. Вкладка Foreign Key таблицы table3

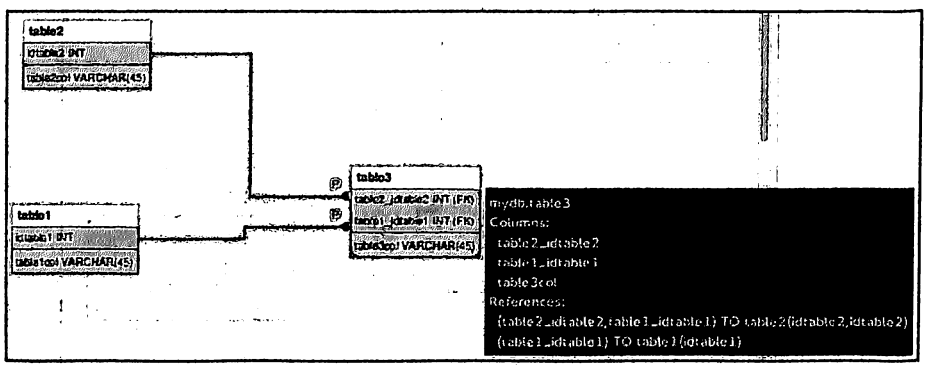


Рис. 4.37. Информация о таблице table3

Если навести курсор мыши на поле с названием таблицы (например, **table3**), то в открывшемся рядом окне будет показана полная информация по базе данных (**mydb**), о столбцах и связях с другими таблицами (рис. 4.37).

Рассмотрим еще некоторые области окна редактирования ERR-диаграмм (см. рис. 4.10). Обратите внимание, что в левой области окна в самой верхней части отображается вся ER-диаграмма целиком, а видимая область экрана выделена. При помощи кнопок «+» и «—» можно настроить масштаб изображения (рис. 4.38).

Ниже расположена область навигации, которая позволяет осуществлять переключение между объектами (в данном случае между таблицами 1 и 2 (рис. 4.39). Ниже во вкладке **History** можно посмотреть историю выполненных операций (рис. 4.40).

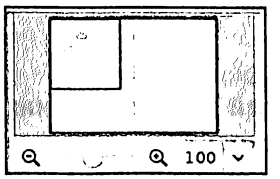


Рис. 4.38. Область для отображения всей ER-диаграммы

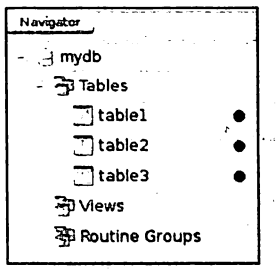


Рис. 4.39. Область навигации

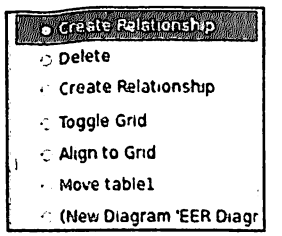


Рис. 4.40. Выполненные операции

Справа в верхнем углу экрана, во вкладке шаблонов (**Templates**) можно выбрать один из имеющихся шаблонов для создания таблиц или создать новую таблицу в разрабатываемой модели на основе шаблона. Также имеется возможность создавать свои собственные модели шаблонов таблиц. Кнопка позволяет открыть редактор шаблонов уже готовых таблиц (Open the table template editor) (рис. 4.41) или создать новый шаблон. Например, при выборе таблицы «Пользователь» (User) будет открыто окно для редактирования, как показано на рис. 4.42.

Для того чтобы добавить готовую таблицу в модель, необходимо выбрать ее в имеющихся шаблонах и нажать на кнопку . Выбранная таблица будет добавлена во вкладку Tables создаваемой модели.

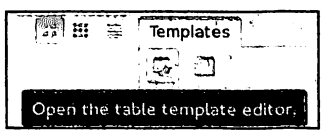


Рис. 4.41. Вкладка шаблонов

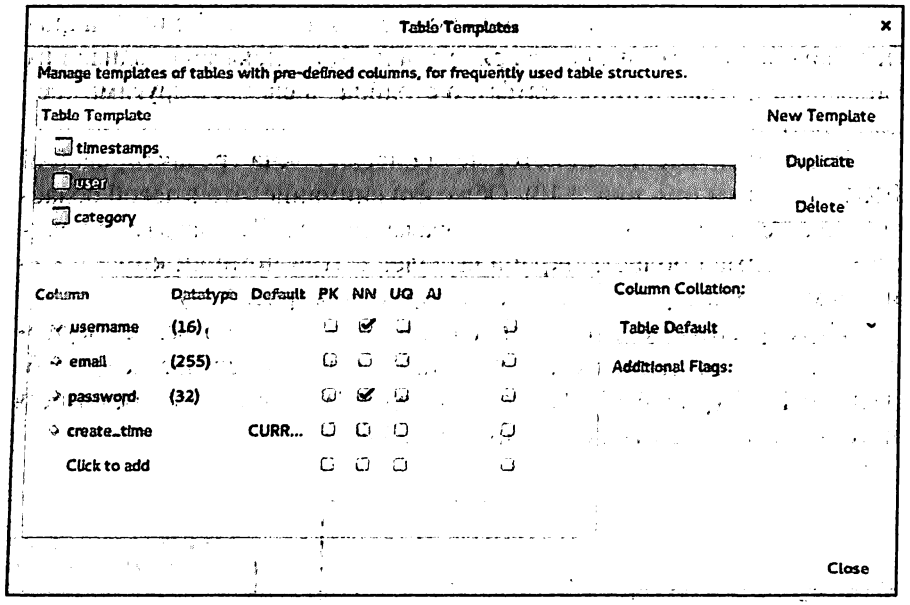


Рис. 4.42. Шаблон таблицы **User**

После детального рассмотрения возможностей проектирования ER-диаграмм в MySQL Workbench можно переходить к созданию диаграмм различной степени сложности.

Учащимся предлагается выполнить практическую работу, основанную на изученном теоретическом материале. Выполнение практической работы предполагает использование **MySQL Workbench** для проектирования ER-моделей.

Порядок выполнения практической работы

1. Ознакомиться с описанием.
2. Открыть рабочую область программы MySQL Workbench для создания ER-моделей. Произвести настройки нотации модели.
3. Создать тестовую ER-модель не менее чем из двух таблиц. В каждой таблице задать первичный ключ и несколько столбцов, выбрать для столбцов различные типы данных.
4. Выбрать тип связи из имеющихся на панели инструментов. Создать выбранную связь между таблицами.
5. Перейти на вкладку шаблонов и создать свой собственный шаблон для таблицы в соответствии с заданием. Сохранить его.

6. Добавить таблицу из шаблонов на ER-диаграмму.
7. Перейти на вкладку MySQL Model и произвести настройки отображения, например, отобразить таблицы в виде списка.
8. Сохранить модель и выйти из программы.

Контрольные вопросы

1. Какие пункты меню имеются в рабочей области **EER Diagram**? Для чего они предназначены?
2. Какие нотации для создания ER-моделей имеются в MySQL Workbench? Каким образом производятся настройки нотаций?
3. Какие виды связей доступны для построения в ER-моделях? Чем они отличаются?
4. Каким образом создается таблица и ее атрибуты в ER-модели?
5. Что такое первичный ключ? Внешний ключ? В какой области таблицы отображается первичный ключ?
6. Опишите процесс построения неидентифицирующей связи один-ко-многим.
7. Для чего нужны шаблоны таблиц? Каким образом их можно создать и использовать?
8. В какой вкладке MySQL Model имеется возможность настроить роли пользователей?
9. Каким образом можно сохранить, а затем использовать созданную ER-диаграмму?

Лабораторная работа 5
ПОСТРОЕНИЕ ER-МОДЕЛЕЙ. ПРЯМОЙ И ОБРАТНЫЙ ИНЖИНИРИНГ

Цель: формирование навыков по исследованию заданной предметной области, умения выделить сущности и атрибуты, создания ER-модели базы данных.

Исследование предметной области

В качестве примера будет рассмотрено построение базы данных для распределения и прохождения студентами практики. Для наглядности построения модели будут сделаны предположения, упрощающие предметную область. В базе данных должны храниться следующие данные:

- данные о конкретном студенте, включая его контактные данные, месте учебы и специальности;
- сведения об организации и отделе, в который студент направляется на практику;
- данные о руководителе практики (для конкретного студента), включая контактные данные руководителя, его должность, организацию, в отделе которой он работает.

Будем считать, что руководитель практики руководит студентами только в той организации и в том отделе, в котором он работает, таким образом, организация и отдел практики студента однозначно определяются по организации и отделу руководителя.

Идентификация сущностей

Основные компоненты ER-диаграммы — это сущности, атрибуты и связи. Каждая сущность является множеством подобных индивидуальных объектов, называемых экземплярами. Каждый экземпляр

индивидуален и должен отличаться от всех остальных экземпляров. Атрибут выражает определенное свойство объекта. С точки зрения базы данных как физической модели сущности соответствует таблица, экземпляру сущности — строка в таблице, а атрибуту — колонка таблицы.

Сущность на диаграмме изображается прямоугольником. В зависимости от режима представления диаграммы прямоугольник может содержать имя сущности и список ее атрибутов. Изображения сущности могут настраиваться, о чем будет сказано ниже.

Создание ER-модели

Для построения ER-модели определим набор сущностей и зададим связи между ними.

Как известно, для выделения сущностей (т. е. основных понятий предметной области, информация о которых подлежит хранению) необходимо сначала описать предметную область. Это может быть сделано различными способами, первоначально осуществим словесное описание.

В первую очередь выделим сущности для описания рассматриваемой предметной области — «Распределение и прохождение студентами практики». Анализ бизнес-процессов для рассматриваемой предметной области показал, что необходимо выделить следующие основные сущности:

- организация;
- отдел;
- должность;
- университет (или вуз);
- факультет;
- специальность;
- практика;
- студент;
- руководитель.

Далее определим атрибуты сущностей. Каждый атрибут хранит информацию об определенном свойстве сущности. Поскольку каждый экземпляр сущности должен быть уникальным, то необходимо наличие атрибута (или группы атрибутов), который идентифицирует сущность и называется первичным ключом.

Первичный ключ (primary key) — это атрибут или группа атрибутов, однозначно идентифицирующая экземпляр сущности.

Атрибуты ключа не должны содержать нулевых значений, поскольку каждый экземпляр сущности должен быть уникальным.

К первичным ключам предъявляются определенные требования. Первичный ключ должен однозначно идентифицировать экземпляр сущности. Первичный ключ должен быть компактен, т. е. удаление любого атрибута из состава первичного ключа должно приводить к потере уникальности экземпляра сущности. Каждый атрибут из состава первичного ключа не должен принимать NULL-значений (например, если принять в качестве первичного ключа номер паспорта, необходимо быть уверенным, что все сотрудники имеют паспорта). Каждый атрибут первичного ключа не должен менять свое значение в течение всего времени существования экземпляра сущности (сотрудник может сменить фамилию и паспорт, поэтому данные потенциальные ключи не могут стать первичными).

В табл. 5.1 приведены основные сущности и атрибуты предметной области, а также приведен тип данных атрибутов. Вообще говоря, при использовании на практике такого рода СУБД требуется более углубленное исследование предметной области и, соответственно, выявление дополнительных атрибутов. Однако для рассматриваемого учебного примера ограничимся только основными сущностями и атрибутами.

Таблица 5.1. **Основные сущности и атрибуты предметной области**

Сущность	Атрибут	Ключ	Тип
Организация	Код организации		Целое
	Наименование		Строка
	Дополнительные данные		Строка
Отдел	Код отдела		Целое
	Наименование		Строка
	Дополнительные данные		Строка
Должность	Код должности		Целое
	Наименование		Строка
	Дополнительные данные		Строка

Окончание табл. 5.1

Сущность	Атрибут	Ключ	Тип
Университет	Код университета		Целое
	Наименование		Строка
	Дополнительные данные		Строка
Факультет	Код факультета		Целое
	Наименование		Строка
	Дополнительные данные		Строка
Специальность	Код специальности		Целое
	Наименование		Строка
	Дополнительные данные		Строка
Практика	Код		Целое
	Приказ		Строка
	Дата начала практики		Дата/время
	Дата окончания практики		Дата/время
	Запись о прохождении практики		Строка
Студент	Код студента		Целое
	ФИО		Строка
	Дата рождения		Дата/время
	Телефон		Строка
	e-mail		Строка
	Дата поступления в вуз		Дата/время
	Дополнительные данные		Строка
Руководитель	Код руководителя		Целое
	ФИО		Строка
	Телефон		Строка
	e-mail		Строка
	Дополнительные данные		Строка

Напомним, что сущность является независимой, если она может быть уникально идентифицирована без определения ее связей с другими сущностями. Зависимая сущность, наоборот, не может быть уникально идентифицирована без определения ее связей с другими сущностями. В данном случае все сущности являются независимыми.

Войдем в режим построения ER-моделей, как было описано ранее.

Будем использовать латиницу для названия сущностей и атрибутов, поэтому используем следующие названия таблиц:

Студент — **student**;
Руководитель — **tutor**;
Организация — **organization**;
Отдел — **department**;
Должность — **position**;
Университет — **university**;
Факультет — **faculty**;
Специальность — **speciality**;
Практика — **traineeship**.

Приступим к построению ER-модели, для чего поместим на рабочую область построения ER-моделей сущность (таблицу) **Организация** (рис. 5.1).

Имя таблицы задается в соответствующем поле вкладки **Table** (см. рис. 5.1, внизу). Для создания атрибутов (столбцов) следует перейти на вкладку **Columns** (рис. 5.2). Во избежание проблем с настрой-

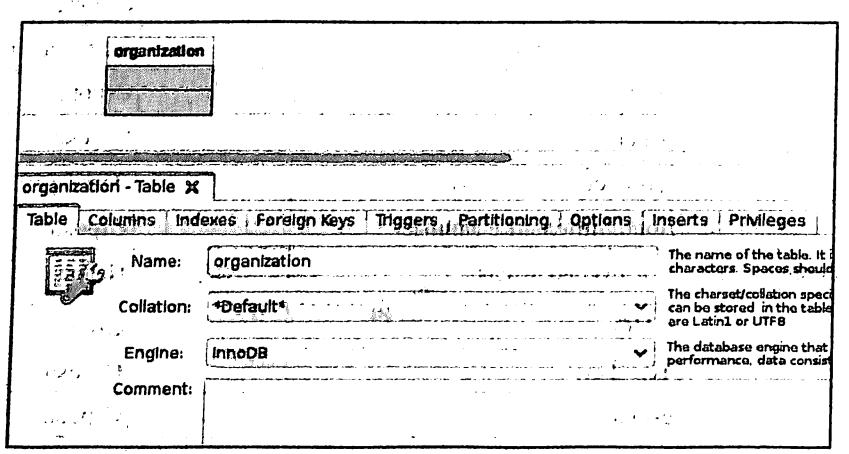


Рис. 5.1. Создание новой таблицы **organization**

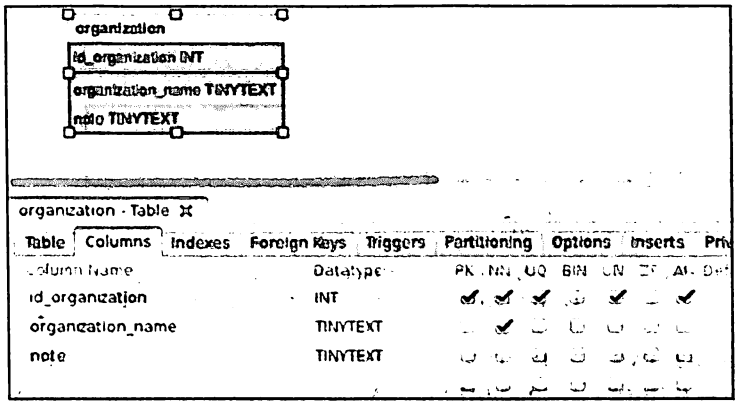


Рис. 5.2. Внесение атрибутов в таблицу **organization** (Организация)

ками кириллицы все названия таблиц и столбцов следует давать на латинице. В качестве значений по умолчанию можно задать **NULL** — неопределенное или внести то значение, которое будет автоматически подставлено в незаполненное поле. Также поле можно оставить и незаполненным.

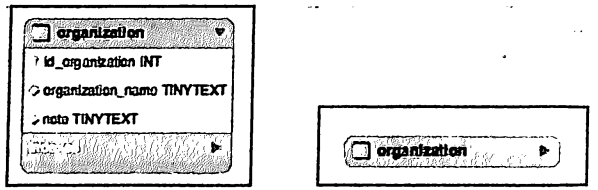
Атрибуты первичного ключа на диаграмме помечены специальным значком ключа. Он появляется тогда, когда в редакторе во вкладке **Columns** в соответствующей опции проставляется галочка.

Для единообразного названия атрибутов принимаются следующие наименования:

- первичный ключ — **id**+краткое название таблицы;
- наименование — название таблицы_**name**;
- дополнительные данные — **note**.

Внесем все выявленные атрибуты (см. табл. 5.1) в соответствующей вкладке (см. рис. 5.2). Для названия организации потребуем, чтобы данное поле имело значение **NOT NULL**, т. е. поле должно содержать какие-либо данные, поскольку любая организация имеет название. Аналогично можно потребовать, чтобы поля ФИО (фамилия, имя, отчество), название университета, факультета, специальности, отдела и должности также имели значение **NOT NULL**.

В некоторых версиях **MySQL Workbench** имеется возможность скрыть атрибуты на диаграмме, нажав на стрелку, смотрящую вниз рядом с названием таблицы. В данной версии **MySQL Workbench** это можно сделать только для нотации **Workbench Default** (рис. 5.3).



а *б*

Рис. 5.3. Отображение таблицы **organization** с атрибутами (*а*) и без атрибутов (*б*) в нотации **Workbench Default**

Анализируя предметную область, отметим, что таблицы **student** (Студент) и **tutor** (Руководитель) имеют несколько одинаковых полей. Создадим для них шаблон под названием **person**. Для этого перейдем в редактор для создания шаблонов и нажмем левую кнопку, а в открывшемся окне — кнопку «New Template». Зададим имя **person** (рис. 5.4).

В соответствии с выявленными атрибутами предметной области создадим столбцы таблицы, потребовав, чтобы поле, в котором содержится ФИО, было **NOT NULL** (рис. 5.5). Заметим, что в данном случае для хранения фамилии, имени и отчества выделяется одно поле. В некоторых случаях, когда речь идет о хранении большого ко-

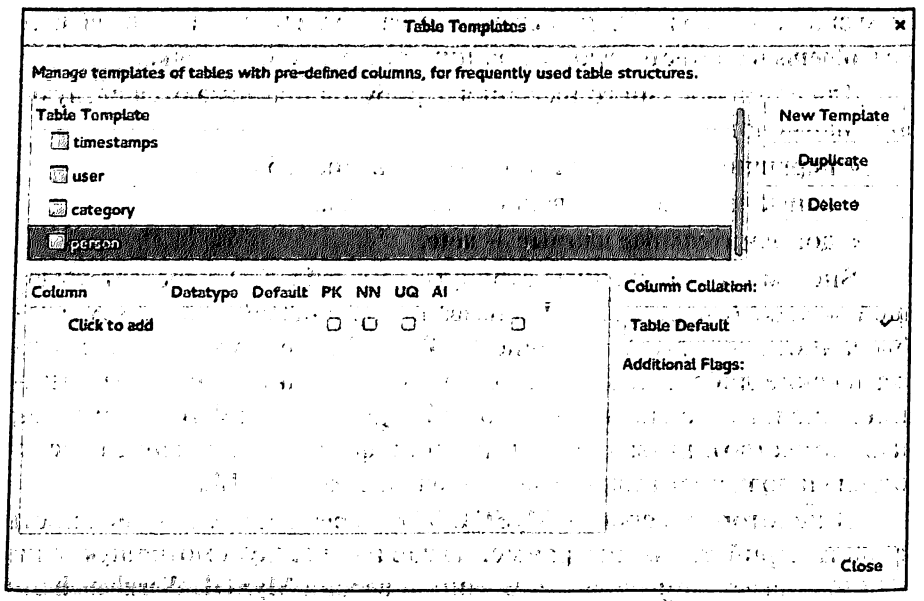


Рис. 5.4. Создание шаблона **person**

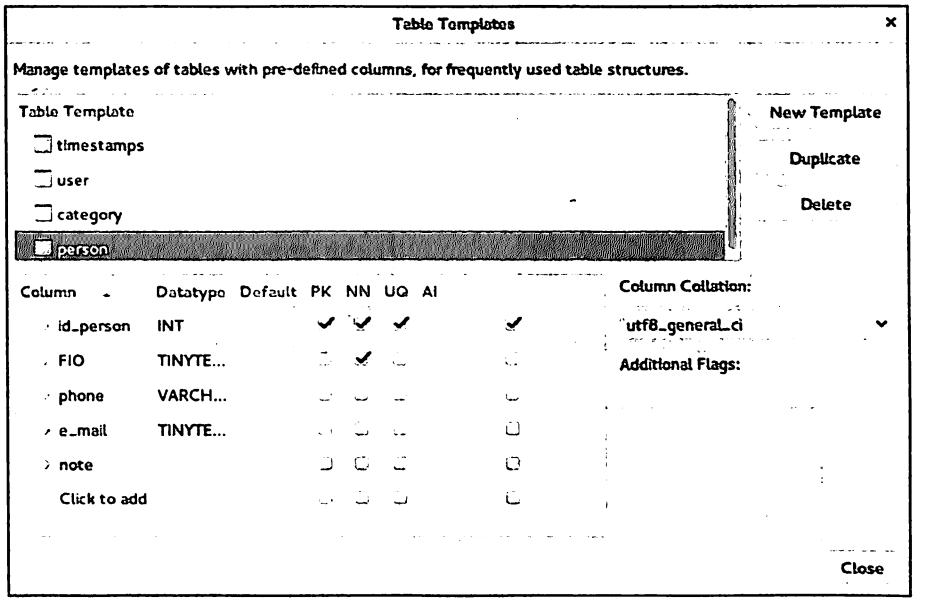


Рис. 5.5. Столбцы шаблона **person**

личества фамилий и имен, среди которых много повторяющихся значений, например, в телефонных справочниках, это поле в таблице логично заменить на три таблицы в СУБД. Это значительно экономит объем хранимой информации, поскольку хранение внешнего ключа (целое число) занимает значительно меньший объем по сравнению с хранением текстовой информации. Ниже будет показано, каким образом это реализуется. Однако в данном примере можно предположить, что хранение ФИО в одном поле допустимо.

Для того чтобы впоследствии данные корректно отображались (в том числе и на кириллице), следует также выбрать кодировку для столбцов **utf8_general_ci**.

Сохраним шаблон, который будет добавлен к имеющимся (рис. 5.6). Если в дальнейшем понадобится в шаблон внести изменения, то в окне шаблонов выбирается нужный и по правой кнопке вызывается контекстное меню (рис. 5.7), в котором выбирается пункт Edit Template, и шаблон можно отредактировать.

На основании этого шаблона можно создать таблицу **student** (Студент), добавив необходимые атрибуты и переименовав первичный ключ (рис. 5.8).

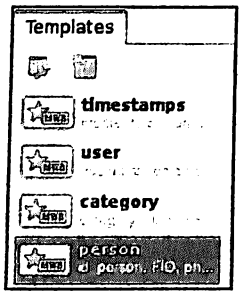


Рис. 5.6. Добавление шаблона **person**

Рис. 5.7. Контекстное меню для шаблона **person**

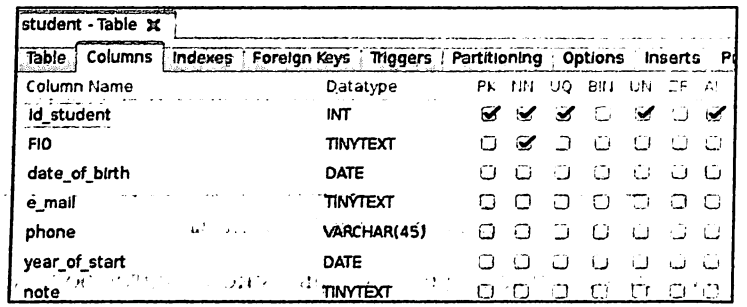


Рис. 5.8. Создание таблицы **student** на базе шаблона **person**

Аналогичным образом создается таблица **tutor** (Руководитель). После этого создадим и все остальные таблицы базы данных.

Последовательно помещаем на диаграмму таблицы в соответствии с исследованием предметной области. На рис. 5.9 представлен окончательный вид внесенных на ER-диаграмму таблиц.

Не забудьте по мере построения модели периодически сохранять результаты работы. Первоначально, чтобы сохранить модель, необходимо выбрать пункт меню **Save as**, затем следует выбирать просто **Save**.

Далее, если не были произведены настройки сущностей и связей, чтобы они отображались в нотации **IDEF1X**, то их необходимо произвести. Выше было сказано, что настройки производятся через выбор пунктов **меню Model → Object Notation → IDEF1X** для сущностей и **Model → Relationship Notation → IDEF1X** для связей. В результате выбора такого вида настроек модель примет вид, как показано на рис. 5.9. Обратите внимание, что независимые сущности в соответствии с нотацией **IDEF1X** имеют вид прямоугольника (зависимые, как было сказано ранее, имеют вид прямоугольника со скругленными углами). Также напоминаем, что ключевые атрибуты находятся в верхней области прямоугольника и отделены от остальных атрибутов горизонтальной чертой. Эта область называется областью ключевых атрибутов.

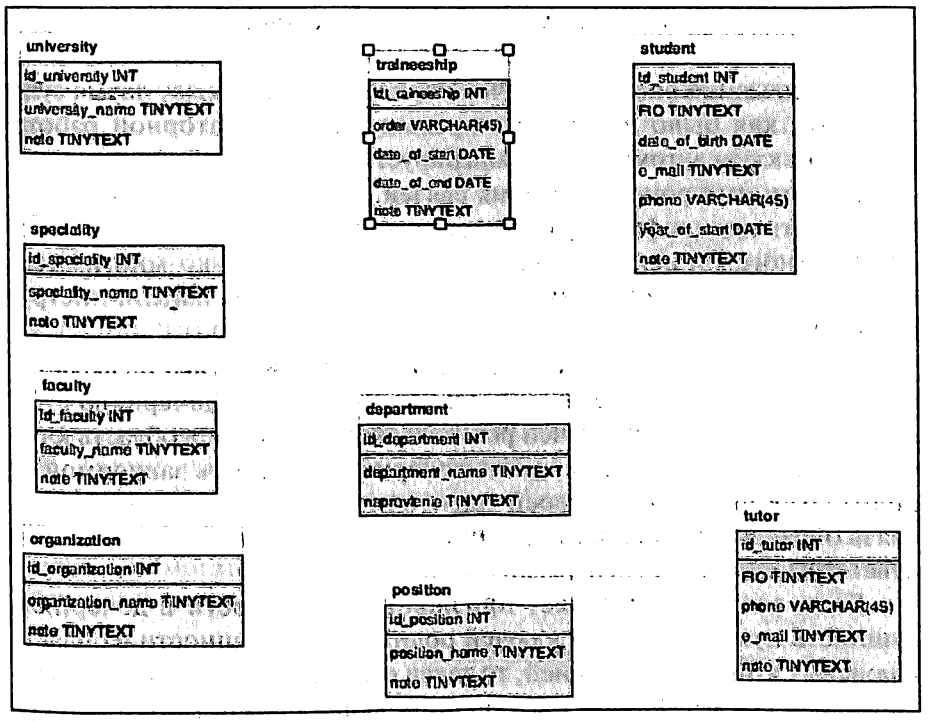


Рис. 5.9. Окончательный вид внесенных на ER-диаграмму таблиц

После завершения создания сущностей можно приступить к созданию связей. Напомним, что при построении ER-модели, когда создается связь между двумя сущностями, **MySQL Workbench** производит миграцию ключевых атрибутов родительской сущности в дочернюю сущность, где они становятся внешними ключами. Поэтому рекомендуется добавлять первичные ключи в независимую сущность сразу же, как только она создана, но создавать внешние ключи для дочерних сущностей нет необходимости, они будут автоматически сгенерированы в процессе построения связей. После того как связь будет создана, **MySQL Workbench** произведет автоматическую миграцию внешних ключей. Кроме того, если удаляется связь, **MySQL Workbench** автома-

тически удаляет соответствующие внешние ключи из дочерних сущностей.

Для создания внешних ключей нарисуйте связь между двумя сущностями, как было показано в предыдущей лабораторной работе. Внешний ключ автоматически мигрирует.

Напомним процесс создания связей. Между сущностями можно установить идентифицирующую связь один-ко-многим, связь многие-ко-многим и неидентифицирующую связь один-ко-многим (соответственно это пиктограммы слева сверху вниз на панели инструментов).

Как было сказано ранее, **идентифицирующей связью** называется связь, которая добавляет признаки идентичности в дочернюю сущность путем миграции ключей родительской сущности в область ключевых атрибутов дочерней, делая дочернюю сущность зависимой от родительской в смысле своей идентичности. В ER-моделях такая связь обозначается сплошной линией с жирной точкой на конце, соответствующем дочерней связи.

Если связь не добавляет признаки идентичности в дочернюю сущность путем миграции ключей родительской сущности в область ключевых атрибутов дочерней, то такая связь называется **неидентифицирующей связью**. При неидентифицирующей связи атрибуты первичного ключа родительской сущности мигрируют в область данных дочерней сущности. В ER-моделях такая связь обозначается пунктирной линией с жирной точкой на конце, соответствующем дочерней связи.

Внешние ключи (Foreign Key) создаются автоматически, когда связь соединяет сущности: связь образует ссылку на атрибуты первичного ключа в дочерней сущности и эти атрибуты образуют внешний ключ в дочерней сущности (миграция ключа). Атрибуты внешнего ключа обозначаются символом (FK) после своего имени.

Поскольку в нашей предметной области все сущности являются независимыми (т. е. уникальность каждой сущности определяется своим уникальным идентификатором — первичным ключом), то связь между такими сущностями является **неидентифицирующей**. Далее необходимо определить мощность (cardinality) связи. Если связь имеет мощность **один-к-одному**, то чаще всего это говорит о том, что база данных не находится в третьей нормальной форме. Если связь имеет мощность **многие-ко-многим**, то с точки зрения нормализации и последующей практической работы с СУБД предпочтительнее заменить ее на две связи **один-ко-многим** и вспомогательную таблицу (см. ниже). Для построения связи, как было сказано ранее, следует выбрать на панели управления соответствующую связь, произвести щелчок мыши по таблице, которая является дочерней, а затем по родительской таблице. Для того чтобы решить, какая сущность является родительской, а какая дочерней, достаточно определить, какая сущность от какой находится в зависимости.

Прежде чем строить связи в рассматриваемой модели, приведем еще некоторые соображения, касающиеся структуры базы данных. Очевидно, что сущности **Университет** и **Факультет** связаны соотношением **многие-ко-многим**. Это следует, например, из того, что в университете существует много факультетов, но также факультет с одинаковым названием может присутствовать в различных университетах. Таким образом, если необходимо установить однозначное соответствие между университетом и факультетом (т. е. установить связь **один-ко-многим** вместо связи **многие-ко-многим**), то необходимо ввести вспомогательную таблицу **university_faculty**. Создадим таблицу и ключевое поле (рис. 5.10).

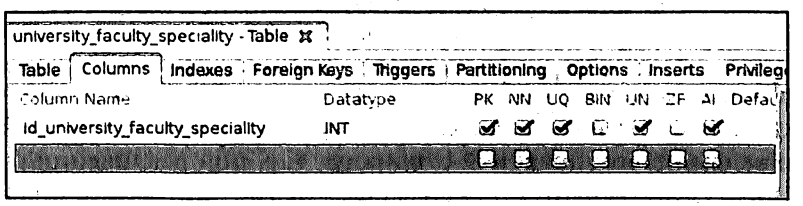


Рис. 5.10. Таблица **university_faculty**

После того как будут построены связи (очевидно, что данная таблица будет являться дочерней для таблиц **Университет** и **Факультет**, поскольку является вспомогательной), внешние ключи этих таблиц мигрируют. Окончательный вид этой таблицы в нотации **IDEF1X** представлен на рис. 5.11, а структура столбцов — на рис. 5.12.

Обратите внимание на то, что у сущности **university_faculty** добавились атрибуты первичного ключа от двух сущностей **university** и **faculty**. Эти атрибуты помечены буквами **FK**. Говорят, что атрибут **мигрировал**, а **FK (Foreign Key)** означает, что атрибут является частью внешнего ключа. Для идентифицирующей связи внешний ключ всегда входит в первичный ключ дочерней сущности, для неидентифицирующей — не входит.

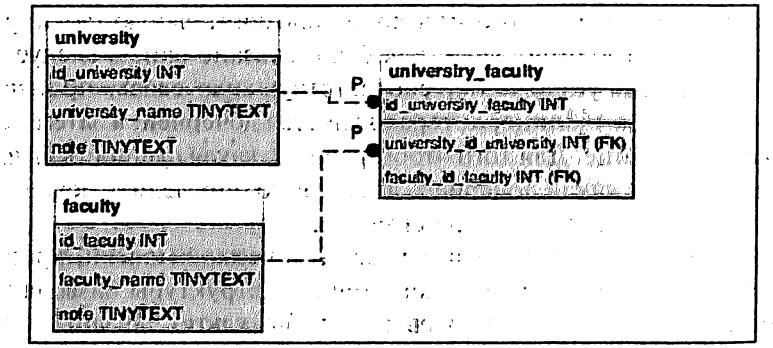


Рис. 5.11. Таблица **university_faculty** после построения связей

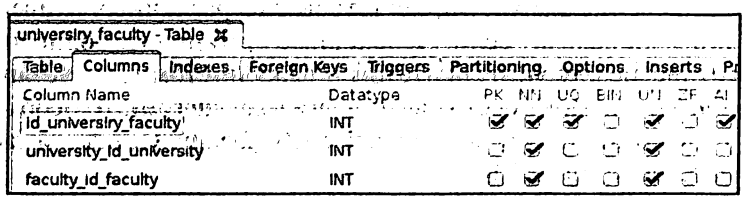


Рис. 5.12. Вкладка **columns** таблицы **university_faculty** после построения связей

Мощность связи отображается на модели буквой **P** и означает, что каждый экземпляр родительской сущности связан с одним или более экземпляром дочерней сущности, т. е. при реализации в конкретной СУБД тип отношения между этими двумя таблицами — **один-ко-многим**.

Наведя мышь на таблицу, можно увидеть свойства атрибутов (рис. 5.13). В то же время, перейдя на вкладку **Columns**, посмотреть, какие добавились столбцы (внешние ключи).

Если перейти на вкладку **Foreign Key** таблицы **university_faculty**, то можно увидеть всю информацию по внешним ключам и произвести настройки (рис. 5.14). В частности, произвести настройки опций ссылочной целостности для изменения и удаления связи в правом окне вкладки (рис. 5.15).

Установки ссылочной целостности определяют, какие действия должна выполнить СУБД при удалении, вставке или изменении строки таблицы (экземпляра сущности). Заданные таким образом действия могут использоваться впоследствии при автоматической генерации триггеров, поддерживающих целостность данных.

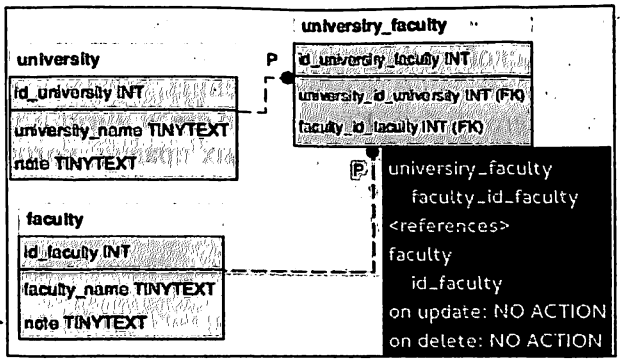


Рис. 5.13. Свойства таблицы **university_faculty** после построения связей

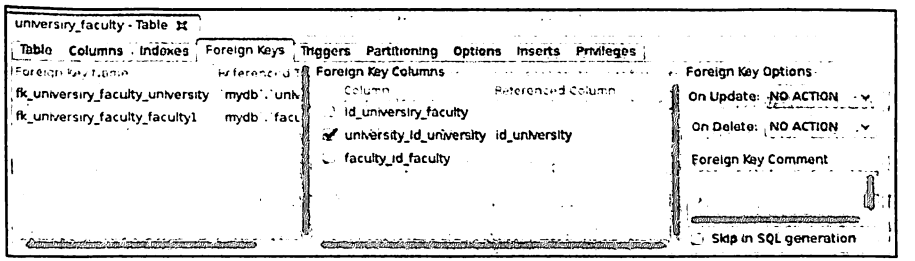


Рис. 5.14. Вкладка **Foreign Key** таблицы **university_faculty**

Существуют следующие виды действий (правил), определяемых в логической модели:

- **RESTRICT** — запрет удаления, вставки или изменения экземпляра сущности;
- **CASCADE** — при удалении экземпляра родительской сущности удаление всех экземпляров дочерней сущности, ссылающихся на удаляемый родительский экземпляр;
- **SET NULL** — при удалении экземпляра родительской сущности атрибутам внешнего ключа всех экземпляров дочерней сущности присваивается значение **NULL**;
- **NO ACTION** — никаких действий не предпринимается.

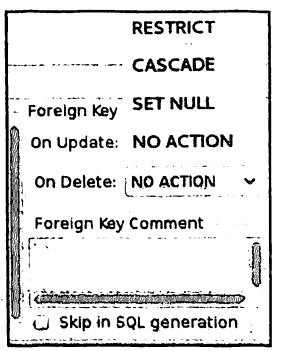


Рис. 5.15. Настройки опций для ссылочной целостности

Эти правила задаются на вставку, удаление и изменение экземпляра как родительской, так и дочерней сущности. Правила, присваиваемые связи по умолчанию, можно изменить, выбрав нужное значение из выпадающего списка. Каждый тип связи имеет в зависимости от вида действия свой набор допустимых правил, который можно найти в соответствующей литературе.

Вкладка **Inserts** служит для внесения данных в таблицу (рис. 5.16).

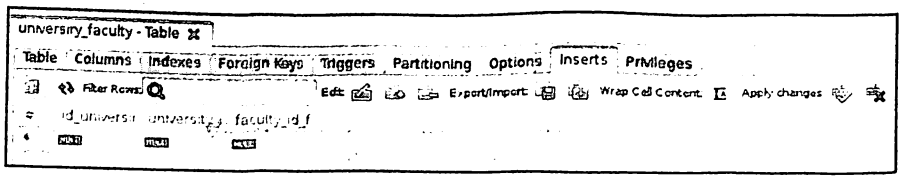


Рис. 5.16. Вкладка **Inserts**

Далее, следуя аналогичным рассуждениям, рассмотрим таблицу **Специальности**. Очевидно, что и этом случае имеется связь многие-ко-многим между специальностью и факультетом университета. И в этом случае для установления однозначного соответствия следует также ввести вспомогательную таблицу **university_faculty_speciality**. Как и в предыдущем случае, при создании таблицы создается только ключевое поле (рис. 5.17), а после построения связей внешние ключи мигрируют. Окончательный вид данной таблицы в нотации **IDEF1X** представлен на рис. 5.18, а структура столбцов на рис. 5.19. Обратите внимание, что внешние ключи отмечены как ненулевые (**NN**) уникальные (**UQ**) атрибуты.

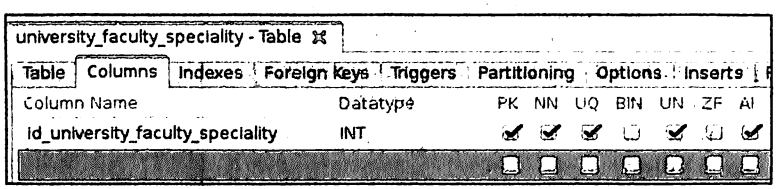


Рис. 5.17. Таблица **university_faculty_speciality**

Теоретически вместо двух вспомогательных таблиц можно было бы сделать одну, связывающую университет, факультет и специальность напрямую. Однако, как показывает практика, факультеты в стабильных университетах меняются редко, поэтому данные этих таблиц будут редко подлежать редактированию. Изменение специально-

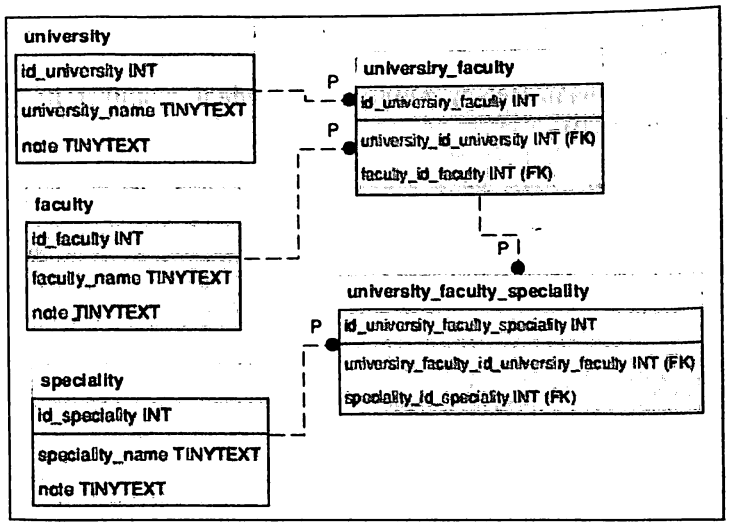


Рис. 5.18. Таблица **university_faculty_speciality** после построения связей

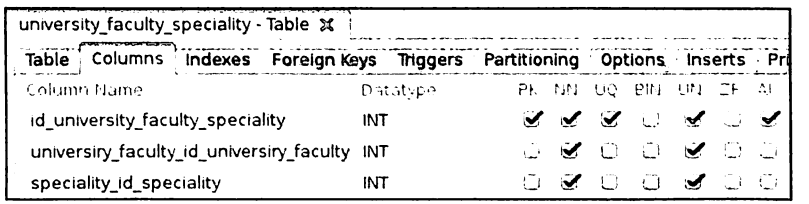


Рис. 5.19. Внешние ключи в таблице **university_faculty_speciality** после построения связей

стей происходит значительно чаще, поэтому такая структура потребует меньшего количества внесения изменений при изменении образовательных стандартов и развития новых направлений подготовки.

Продолжая исследование предметной области, видим, что сущности **Организация**, **Отдел** и **Должность** также связаны соотношением многие-ко-многим. Это следует из того, что в организации существует много отделов, но отдел с одинаковым названием может присутствовать в различных организациях. Аналогичные рассуждения справедливы и для должности. Следовательно, и в данном случае, если необходимо строго соотнести конкретную должность конкретного отдела данной организации, это следует делать через вспомогательную таблицу.

Рассуждая аналогично, определим родительские и дочерние сущности и построим связи между ними.

Построим связь между таблицами **student** и **university_faculty_speciality** (рис. 5.20).

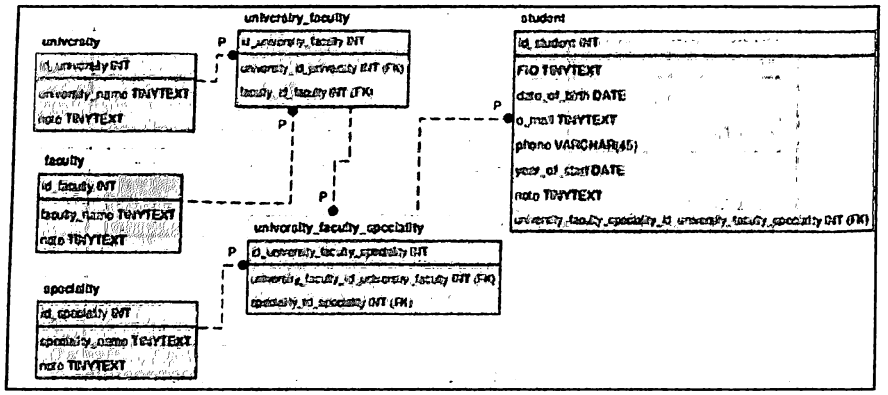


Рис. 5.20. Связь между таблицами **student** и **university_faculty_speciality**

Построим связь между таблицами **tutor** и **organization_department_position** (рис. 5.21).

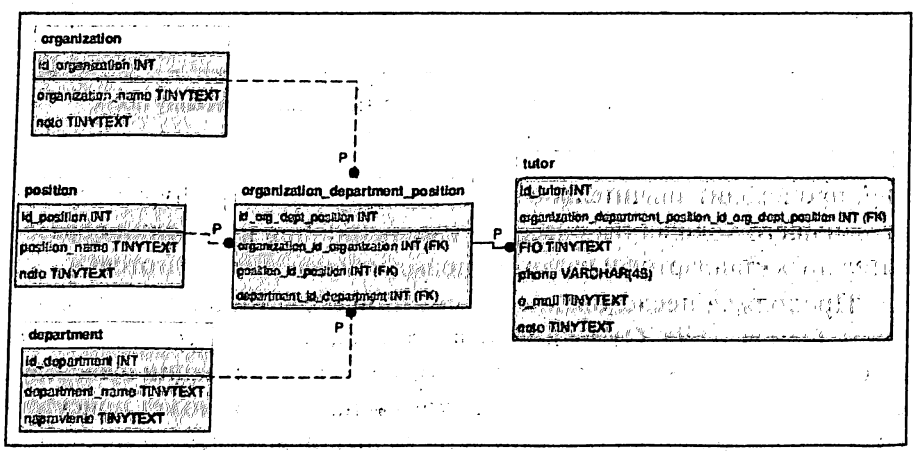


Рис. 5.21. Связь между таблицами **tutor** и **organization_department_position**

Для завершения построения связей необходимо построить связь между таблицами **tutor** — **traineeship** и **student** — **traineeship** (рис. 5.22).

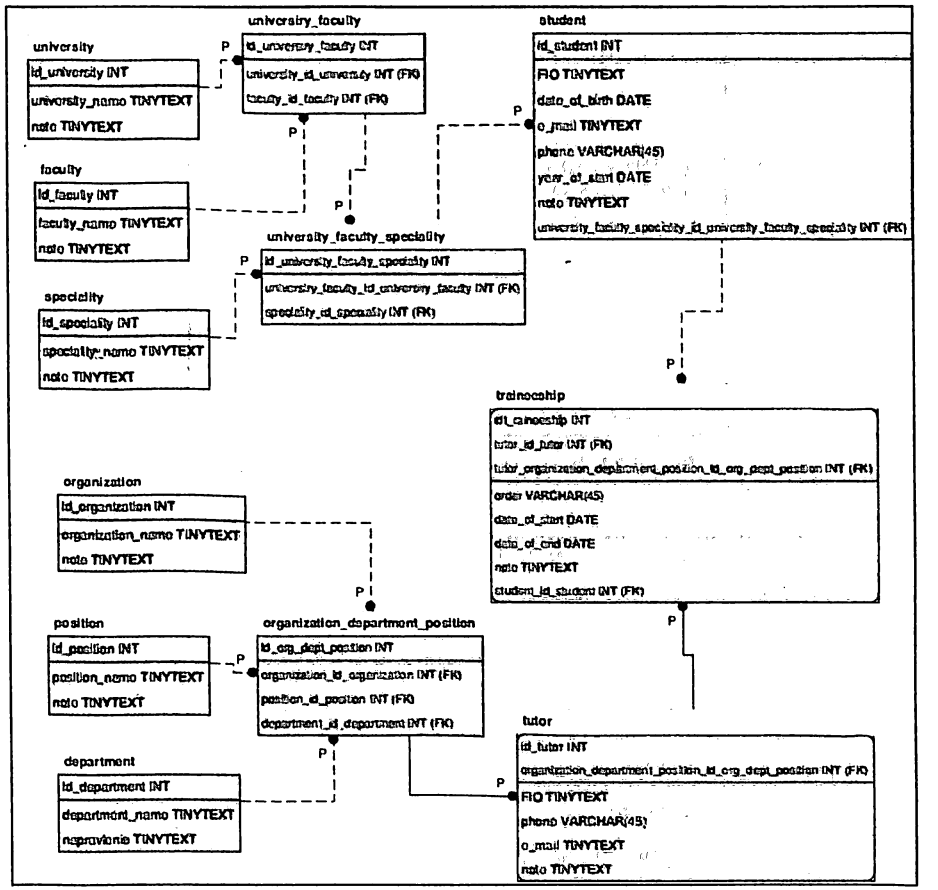


Рис. 5.22. Окончательный вид ER- модели

Создание физической модели (прямой инжиниринг)

Следующий этап проектирования базы данных — переход к физической модели. Для построения реально функционирующей модели необходимо воспользоваться функцией прямого инжиниринга. Как правило, данный процесс не требует никаких настроек, их можно оставить по умолчанию и что-либо изменять, только если пользователь уверен в результатах своих действий. Переход с шага на шаг осуществляется нажатием на кнопку **Next**.

Выберем, как показано на рис. 5.23, меню **Database** → **Forward Engineering**.

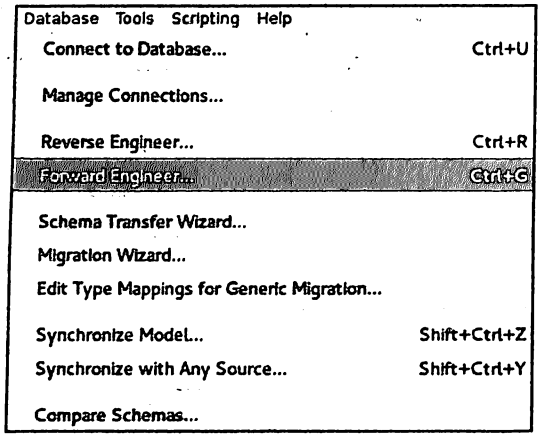


Рис. 5.23. Выбор пункта **Forward Engineering**

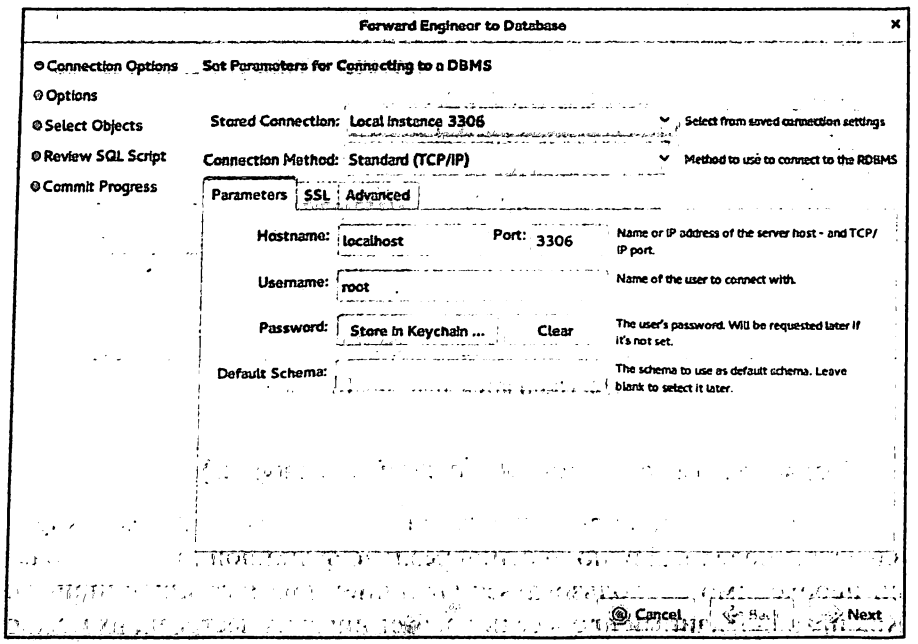


Рис. 5.24. Выбор опций для генерации базы данных

Прямой инжиниринг предназначен для экспорта ER-модели на сервер. Далее будет открыто меню опций, в котором можно выбрать требуемые опции. При наведении на опции курсора возникает всплывающая подсказка, которая поясняет смысл выбираемой опции. Если нет необходимости, можно не выбирать опции, а сразу нажать на кнопку **Next**.

Первый шаг заключается в настройке параметров соединения с сервером базы данных (рис. 5.24). В данном случае следует выбрать соединение с локальным сервером — **localhost**. Также необходимо проверить, что настроен протокол TCP/IP. Остальные настройки, такие как имя пользователя и пароль, производятся по необходимости.

Далее производятся настройки для создаваемой базы данных (рис. 5.25).

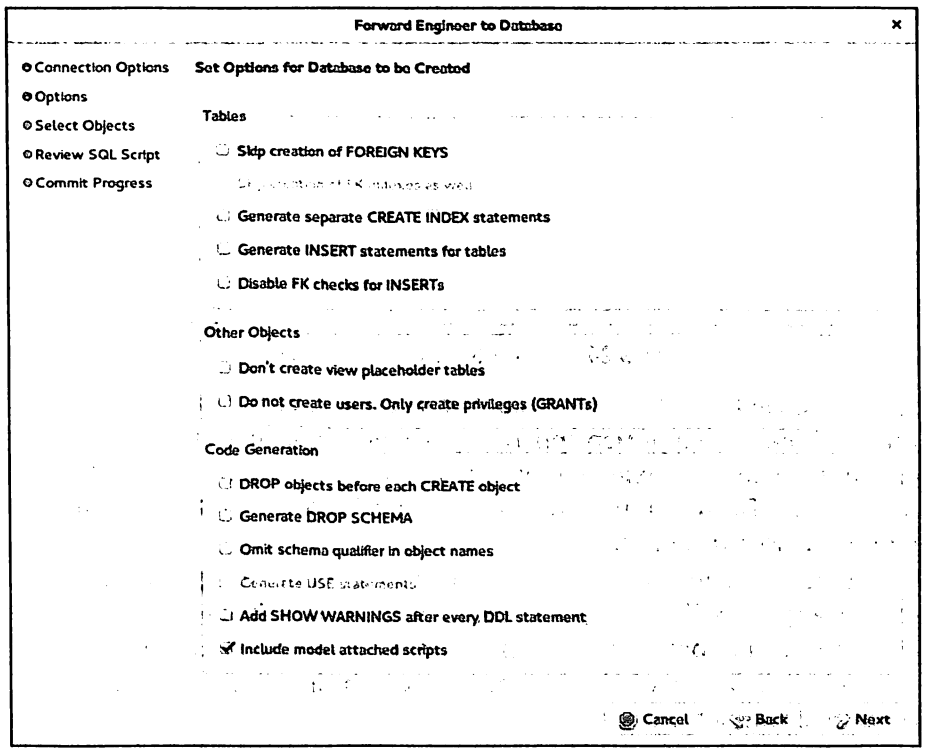


Рис. 5.25. Настройки для создаваемой базы

Следующим шагом является выбор объектов (рис. 5.26). Здесь можно выбрать опции, связанные с объектами, и посмотреть настройки фильтров. В данном случае выбирается только создание объектов — таблиц (12 штук), поскольку никаких других объектов в рам-

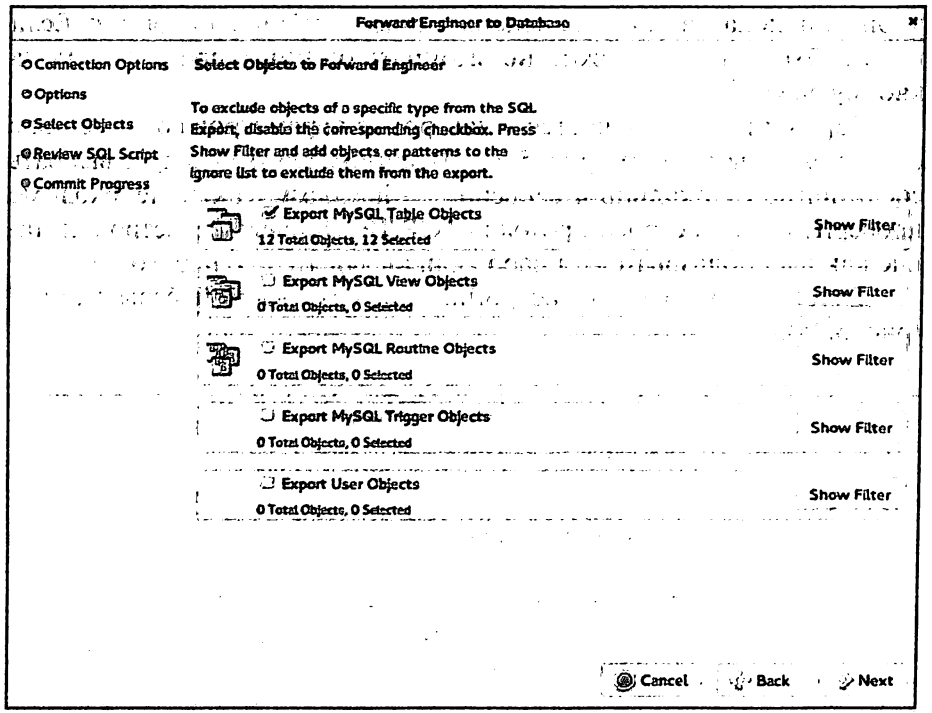


Рис. 5.26. Выбор объектов для генерации

ках ER-модели создано не было. В том случае, если бы были созданы иные объекты, например триггеры, следовало бы поставить галочку в соответствующем поле для их генерации при создании базы данных на сервере. Если при переходе на этот шаг будет требоваться пароль, необходимо его ввести.

Далее можно осуществить предварительный просмотр SQL-скрипта (рис. 5.27), который можно сохранить в файле. Если нет необходимости вносить изменения в сгенерированный код, тогда можно переходить на следующий этап, на котором по данному коду будет создана база данных.

Если никаких изменений вносить не требуется, то для окончания генерации базы данных необходимо нажать кнопку **Next**. После ее нажатия появится окно процесса генерации (рис. 5.28).

Если необходимо посмотреть сообщения (логи), то, нажав на кнопку **Show Logs**, откроем окно **Message Log** (рис. 5.29), в котором будет содержаться информация о процессе генерации кода.

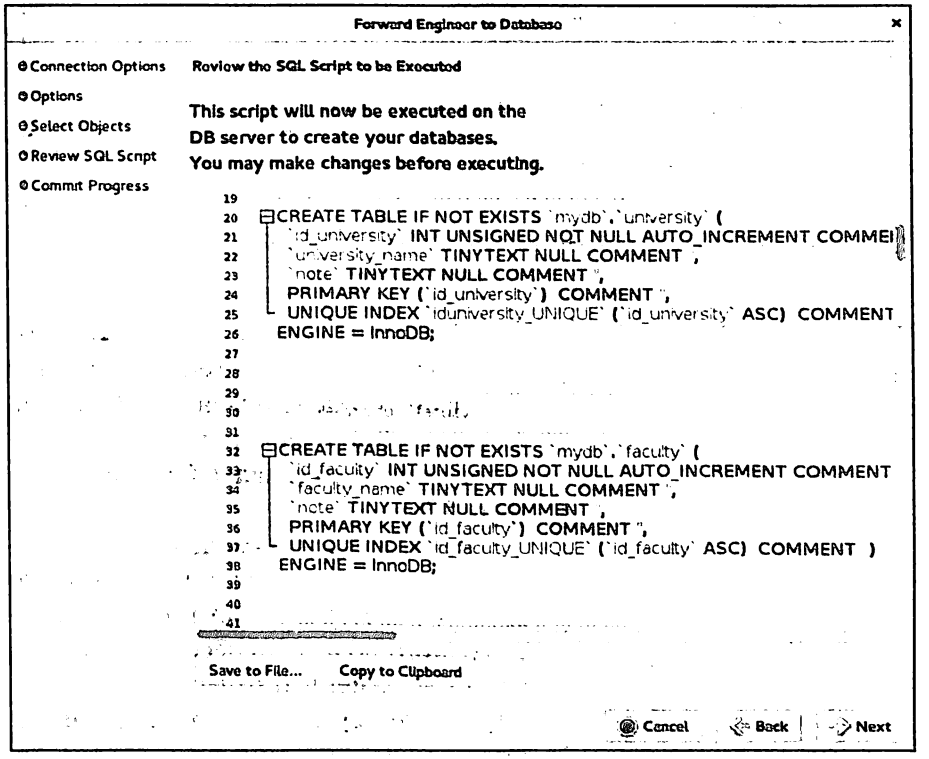


Рис. 5.27. Предварительный просмотр SQL-скрипта

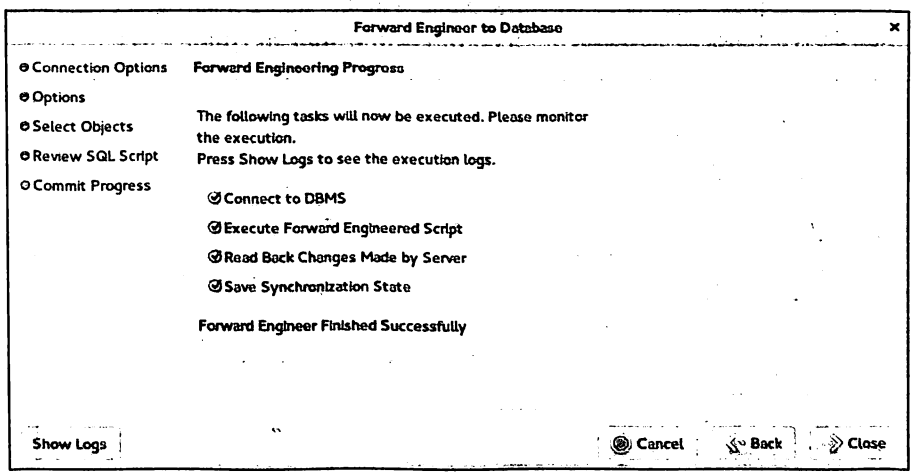


Рис. 5.28. Окно процесса генерации

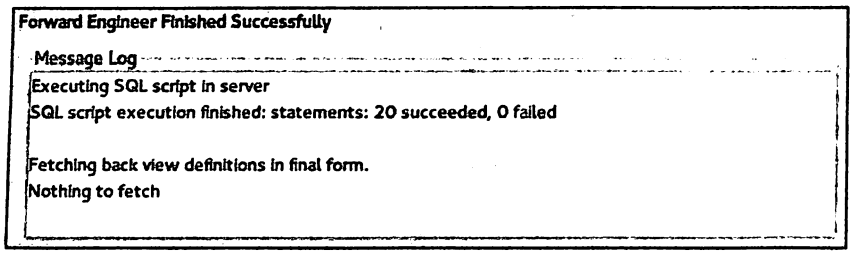


Рис. 5.29. Окно **Message Log**

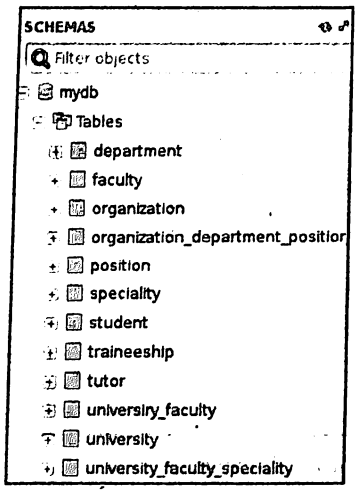


Рис. 5.30. Появление созданной базы данных во вкладках

Нажимаем кнопку **Close** внизу страницы. По завершении генерации новая база данных будет создана из ER-модели. Ее можно будет увидеть во вкладке объектов слева и во вкладке **Overview** внизу (рис. 5.30).

Вернемся к ER-модели и сделаем еще несколько замечаний. Как было сказано ранее, в некоторых случаях вместо одного поля для внесения данных о фамилии, имени и отчестве в таблице **Студент** присутствуют три поля. В эти поля заносятся данные из соответствующих таблиц СУБД. Создадим таблицы **name, middle_name** и **surname**. Свяжем их с таблицей **student** (рис. 5.31).

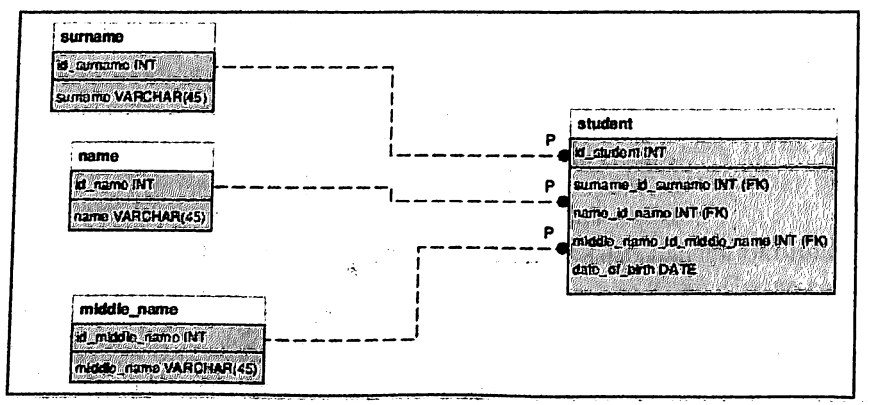


Рис. 5.31. Связывание таблиц **name, middle_name** и **surname** с таблицей **student**

Столбцы таблицы будут выглядеть, как показано на рис. 5.32. При этом поля, являющиеся внешними ключами, будут обязательными для заполнения.

Рассмотрим случай, когда у нас не все поля, связывающие таблицы, обязательны для заполнений, т. е. внешний ключ может иметь неопределенное значение. Это соответствует, например, случаю, ко-

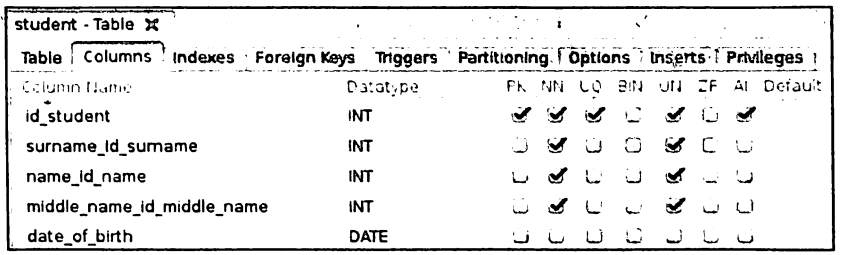


Рис. 5.32. Столбцы таблицы **student**

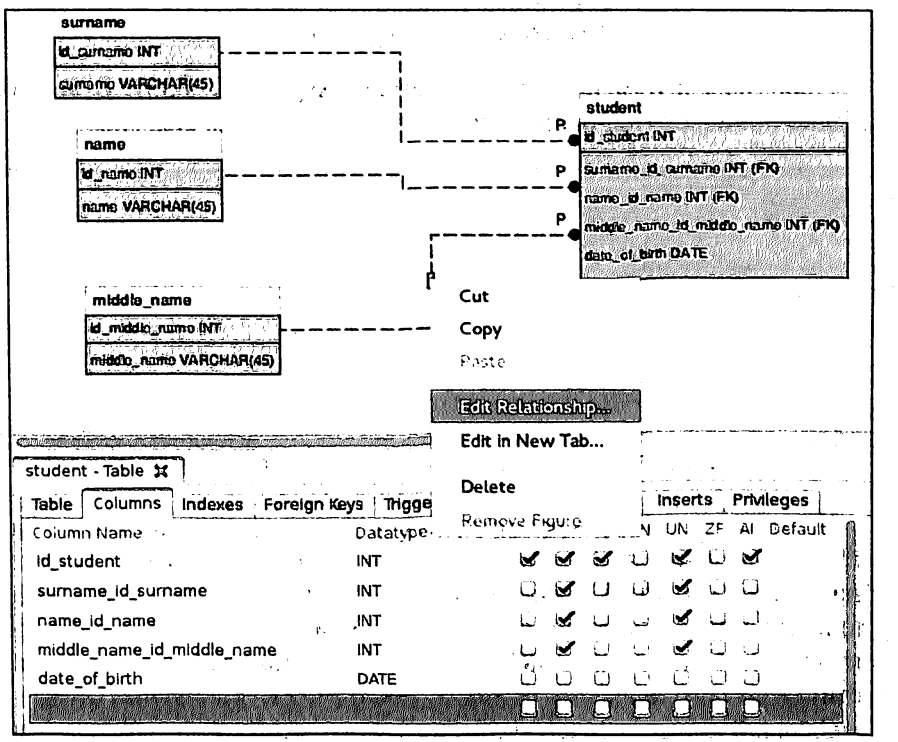


Рис. 5.33. Вызов контекстного меню связей

гда студент может иметь, а может не иметь отчество. В этом случае необходимо несколько изменить вид связи.

Откроем контекстное меню связей, щелкнув правой кнопкой мыши по связи между таблицами **middle_name** и **student** (рис. 5.33).

Далее будет открыто следующее окно редактора связей. У данного окна существуют две вкладки: связи (**Relationship**) и внешние ключи (**Foreign Key**). Во вкладке связей настраиваются свойства, связанные с отображением связей. Также можно дать название связи и написать комментарий (рис. 5.34).

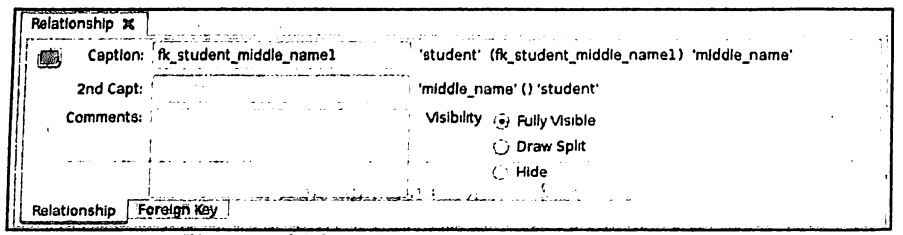


Рис. 5.34. Окно редактора связей

Перейдем в нем на вкладку **Foreign Key** (рис. 5.35). В данной вкладке производятся настройки, связанные с типом связей.

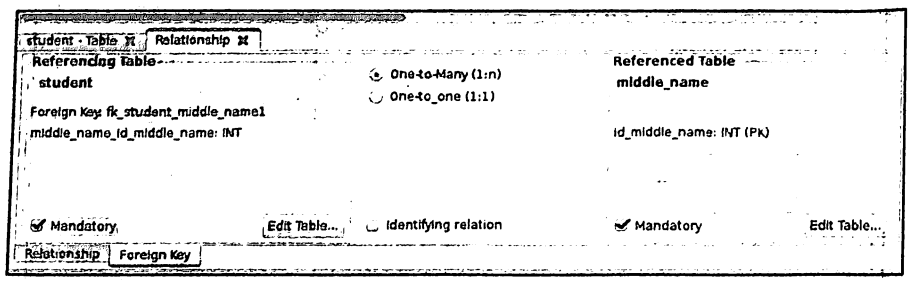


Рис. 5.35. Окно редактора связей, вкладка **Foreign Key**

На этой вкладке есть возможность перейти к редактированию любой из таблиц **student** (кнопка **Edit Table** в левой области), в средней области изменить тип связи (идентифицирующая — неидентифицирующая) и тип отношения (один-к-одному или один-ко-многим), редактированию таблицы **middle_name** (кнопка **Edit Table** в правой области).

Случаю, когда у студента отсутствует отчество, соответствует необязательное наличие значения в поле внешнего ключа соответствующей таблицы. Для этого достаточно снять галочку в правой области (области таблицы **middle_name**) перед словом **Mandatory** (рис. 5.36).

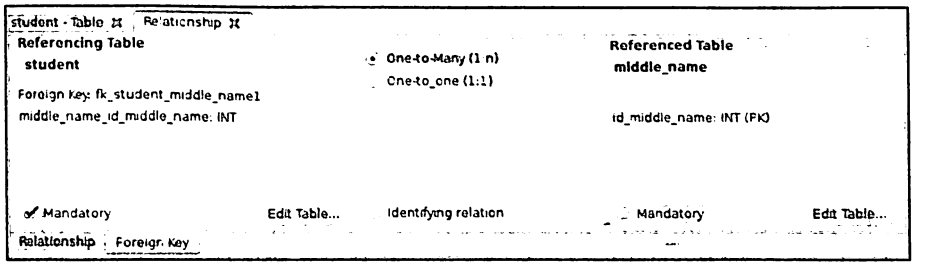


Рис. 5.36. Создание связи, допускающей неопределенные значения

После этого изменится внешний вид связи, представленной на ER-диаграмме (рис. 5.37). В месте присоединения к дочерней таблице там появится ромб, что в нотации **IDEF1X** означает допустимость значений **NULL** для данного поля. Либо, в терминах СУБД, необязательность заполнения данного поля.

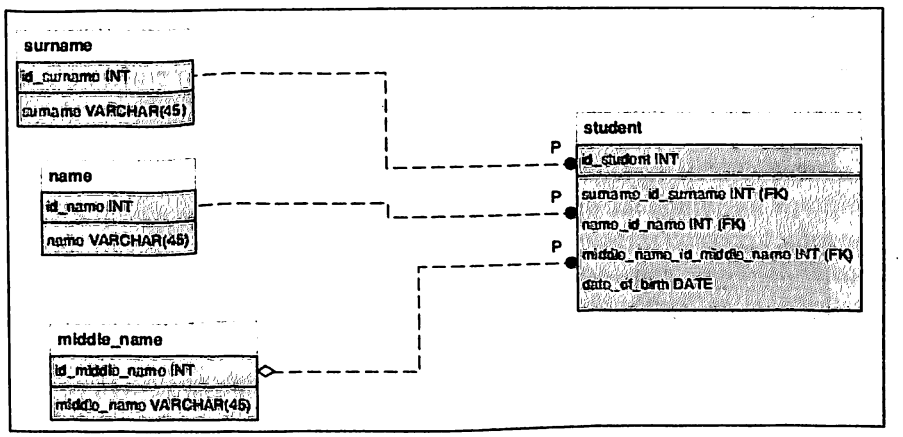


Рис. 5.37. Изменение вида связи

Изменение связи будет отражено во вкладке **Columns** таблицы **student** (рис. 5.38). Галочка в графе **NN (NOT NULL)** внешнего ключа **id_middle_name** будет снята.

Аналогичный результат можно получить, если редактировать таблицу. Верните галочку в правой области (области таблицы **middle_name**) перед словом **Mandatory**. Войдите в режим редактирования таблиц. Откройте таблицу **student**. Измените настройки внешнего ключа

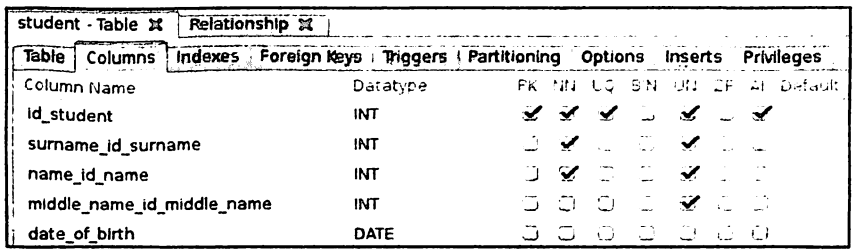


Рис. 5.38. Изменение типа данных внешнего ключа **id_middle_name**

id_middle_name. Выделите соответствующую строку и уберите галочку в столбце NN (NO NULL). В результате этих действий связь изменится и будет выглядеть, как показано на рис. 5.37.

Вернем галочку в правой области (области таблицы **middle_name**) перед словом **Mandatory** и вспомним, каким образом на диаграмме будут отображаться идентифицирующие связи. Для этого войдем в редактор связей и перейдем на вкладку внешнего ключа (см. рис. 5.36).

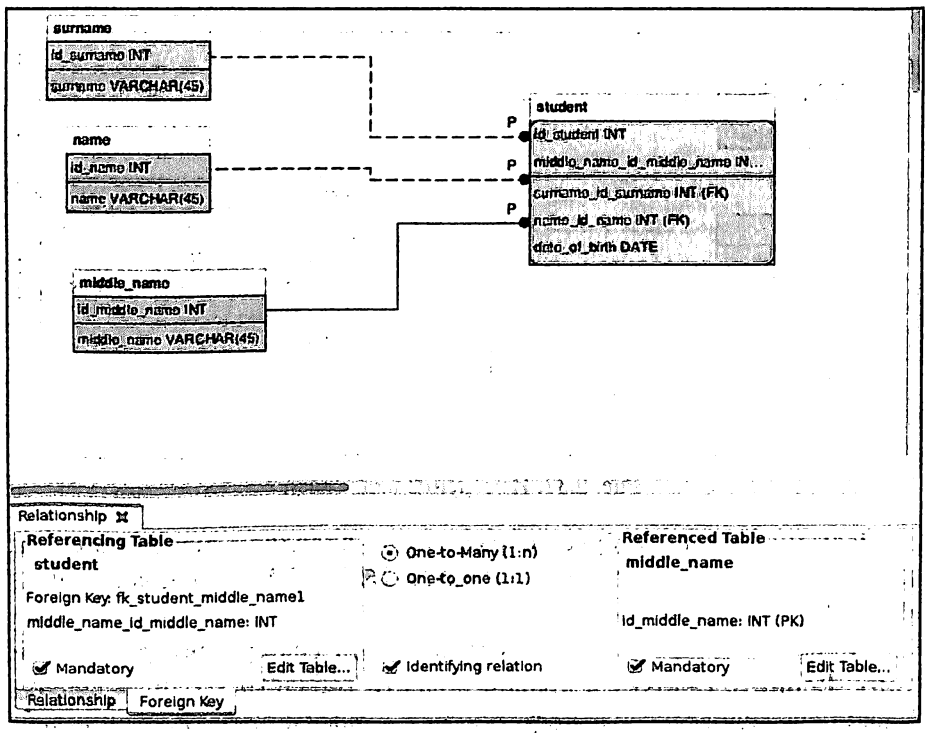


Рис. 5.39. Изменение типа связи между таблицами **id_middle_name** и **student**

Измените настройки типа связи — поставьте галочку перед словами **Identifying relation**. После этого на экране в области отображения ER-диаграммы можно будет увидеть, что теперь связь вместо пунктирной линии отображается сплошной линией. Кроме того, следует обратить внимание на то, что внешний ключ из родительской таблицы поменял свое место: он теперь не просто атрибут, а часть первичного ключа дочерней таблицы. Сама же дочерняя таблица изменила внешний вид и теперь отображается в виде прямоугольника со скругленными углами (рис. 5.39).

Вернитесь к случаю неидентифицирующей связи (снимите галочку).

Для отображения связи один-к-одному в верхней части средней области вкладки внешних ключей выберите соответствующее значение радиокнопки. В результате выполнения этого действия заметьте, что изменилось отображение мощности связи, вместо буквы **P** появилось значение 1, что говорит о представлении связи один-к-одному. Верните настройки связи и сохраните модель.

Обратное проектирование (Reverse engineering)

Обратное проектирование заключается в восстановлении информационной модели по существующей базе данных. Это бывает необходимо при расширении (или модификации) существующей структуры, особенно в тех случаях, когда база данных была создана без необходимой сопроводительной документации. После завершения процесса восстановления ER-модели таблицы автоматически создаются на диаграмме. Теперь можно выполнять модификации уже с использованием логической схемы — добавлять сущности, атрибуты, связи и т. д. По завершении изменений можно синхронизировать модель с базой данных, что актуализирует все проведенные изменения.

Рассмотрим более подробно процесс обратного инжиниринга. Создадим новую модель для ранее созданной базы данных **Student** (рис. 5.40).

Для этого следует вызвать пункт меню **Database** → **Reverse Engineer** (рис. 5.41), после чего откроется окно настроек соединения (рис. 5.42). Настройки производятся так же, как было описано ранее. Переход на следующий шаг осуществляется по нажатию на кнопку **Next**. Кнопка **Show Logs**, как и в случае прямого инжиниринга, открывает окно, которое содержит служебную информацию о ходе процесса инжиниринга.

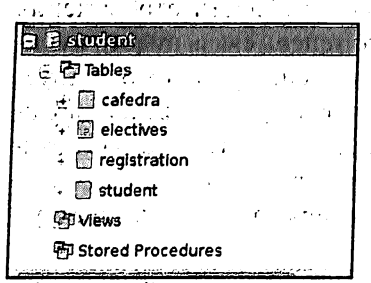


Рис. 5.40. Создание новой модели для базы данных **Student**

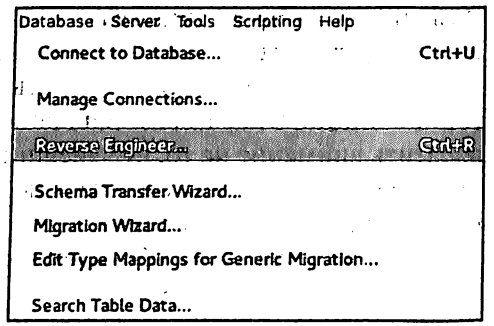


Рис. 5.41. Выбор пункта меню **Обратный инжиниринг**

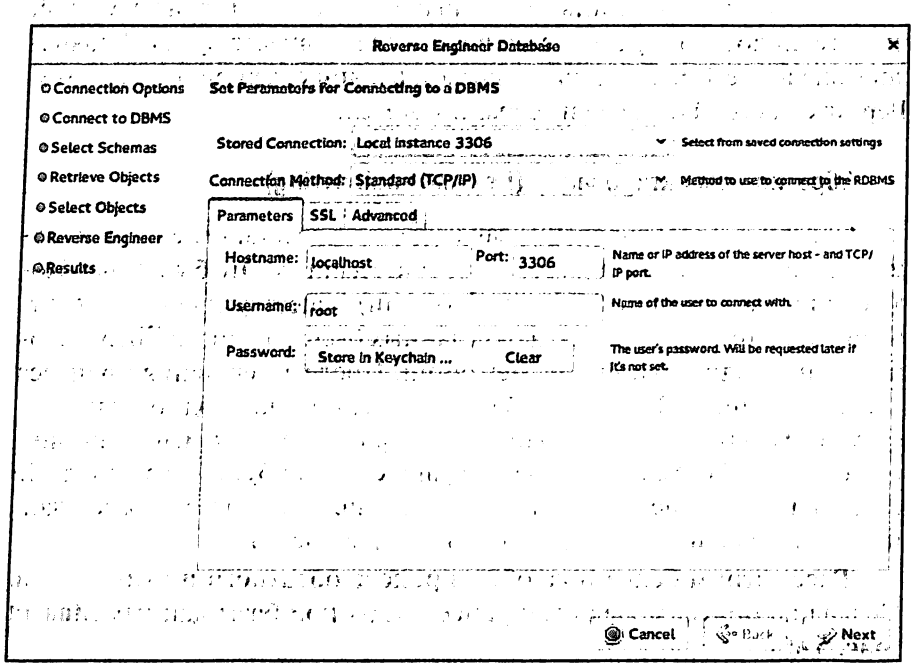


Рис. 5.42. Настройки соединения обратного инжиниринга

После установления соединения откроется окно процесса соединения (рис. 5.43). Когда процесс будет завершен, процессы **Connect to DBMS** и **Retrieve Schema List from Database** будут помечены галочками. В окне **Message Log** (как было сказано ранее) при нажатии кнопки **Show Logs** появятся сообщения об успешном завершении процесса

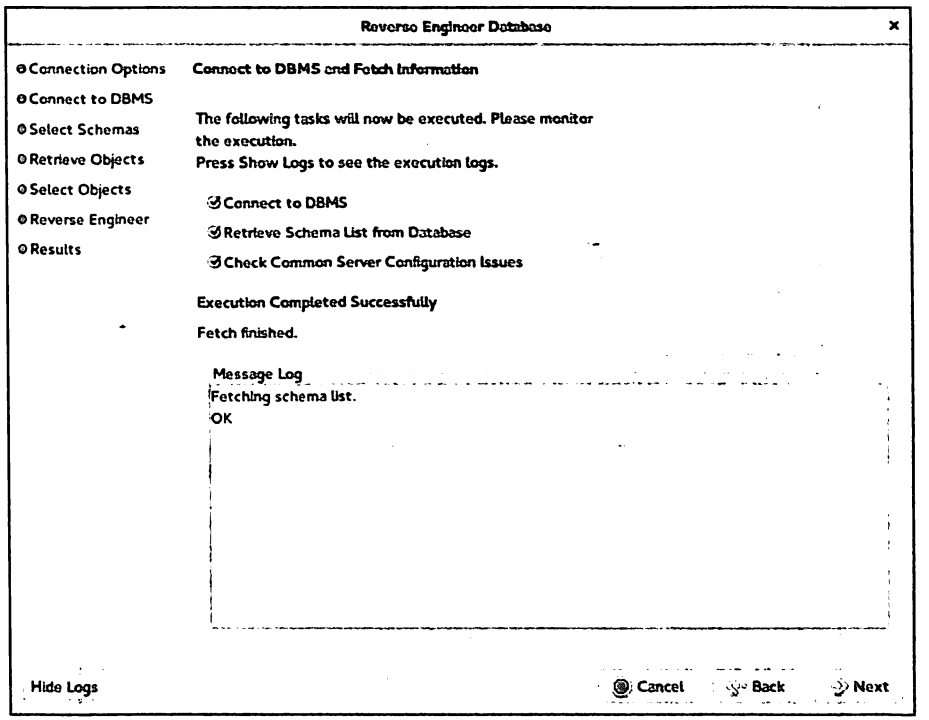


Рис. 5.43. Окно процесса соединения обратного инжиниринга

или возникших проблемах. Если выполнение успешно завершено (**Execution Completed Successfully**), необходимо перейти к следующему шагу, нажав кнопку **Next**.

Следующий шаг состоит в выборе базы данных для обратного инжиниринга (рис. 5.44). Из имеющегося списка баз данных, расположенных на выбранном сервере (в данном случае настройки проведены для сервера **localhost**, поэтому доступны все базы, расположенные на локальном сервере), следует выбрать ту базу данных, обратный инжиниринг которой необходимо сделать. Нажать кнопку **Next** и перейти на следующий шаг.

В следующем окне находится информация по объектам (рис. 5.45).

После нажатия кнопки **Next** открывается окно выбора объектов для обратного инжиниринга (рис. 5.46).

Если есть необходимость генерировать не все таблицы, а только некоторые, то, нажав на кнопку **Show Filter**, можно выбрать, какие именно таблицы будут сгенерированы (рис. 5.47).

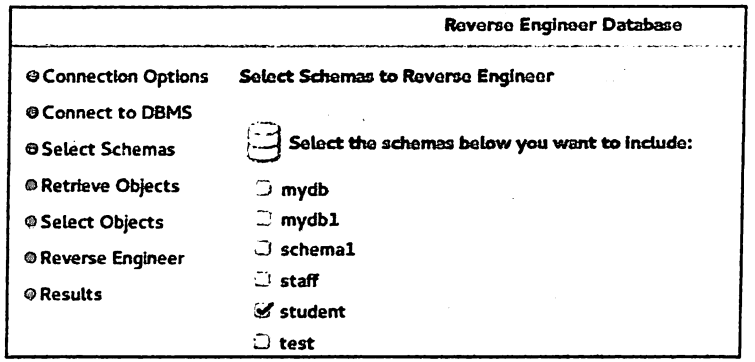


Рис. 5.44. Выбор базы данных для обратного инжиниринга

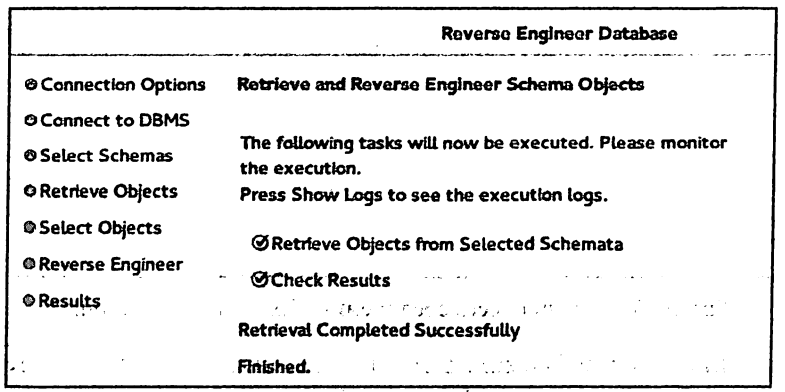


Рис. 5.45. Информация об объектах

Осуществив выбор объектов, переходим на следующий шаг, нажав кнопку **Execute**. По нажатии этой кнопки начинается процесс обратного инжиниринга (рис. 5.48). После появления на экране сообщения об успешном завершении процесса можно нажать кнопку **Next**. Если в процессе обратного инжиниринга возникли проблемы, сообщения о них можно посмотреть в окне **Message Log**, нажав кнопку **Show Logs**.

После завершения процесса обратного инжиниринга появится окно с результатами выполнения (рис. 5.49). В данном случае в окне приведен отчет, в котором указано, что сгенерированы четыре таблицы из базы данных **Student**. Таким образом, процесс создания ER-диаграммы базы данных прошел успешно. Нажав на кнопку **Close**, закрываем окно. В результате выполнения обратного инжини-

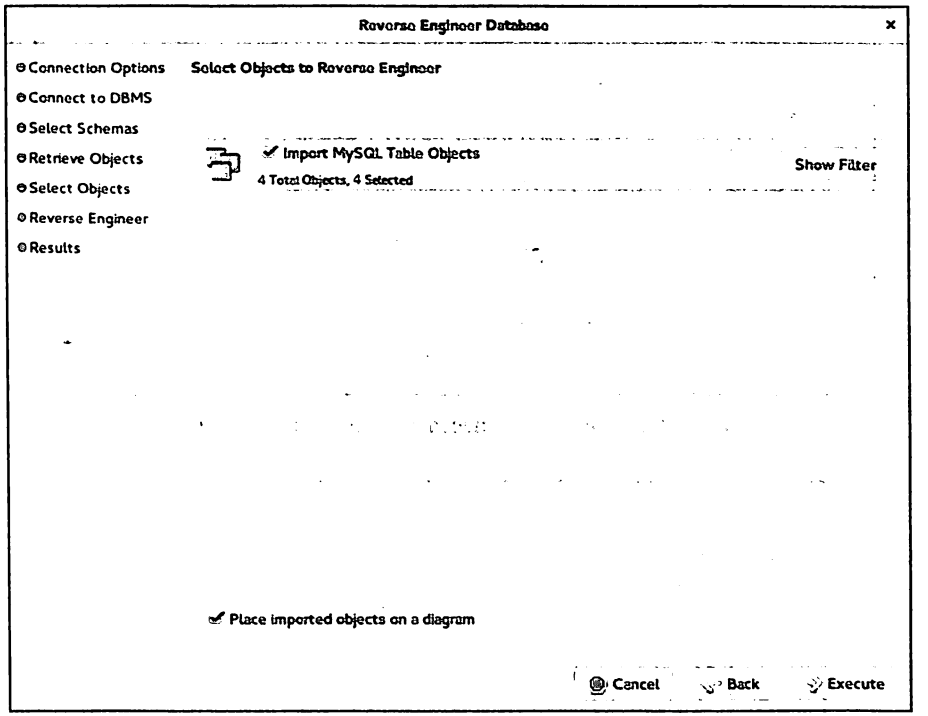


Рис. 5.46. Окно выбора объектов для обратного инжиниринга

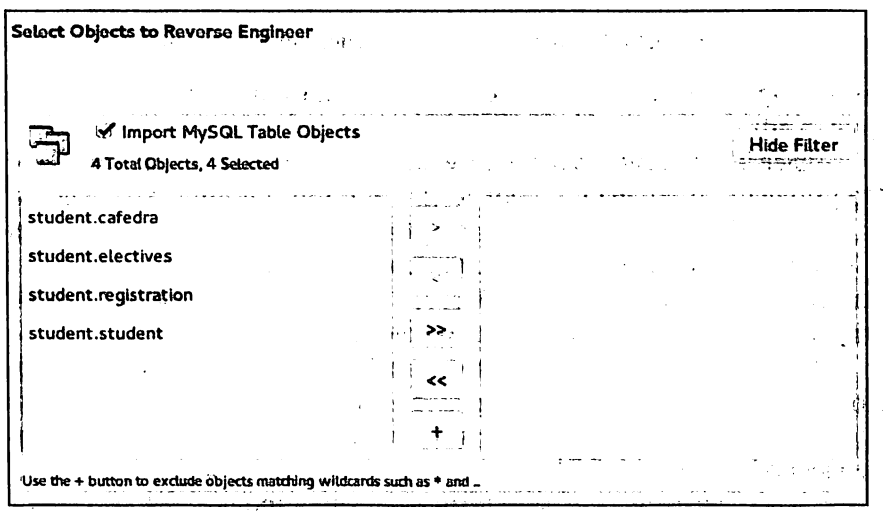


Рис. 5.47. Выбор таблиц для генерации

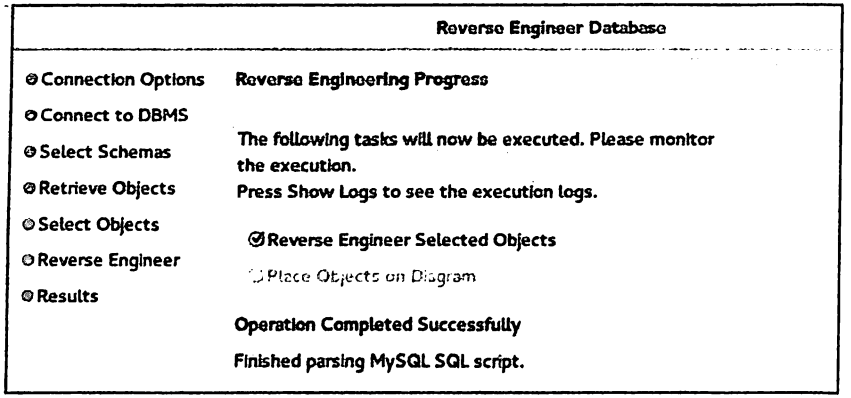


Рис. 5.48. Генерация процесса обратного инжиниринга

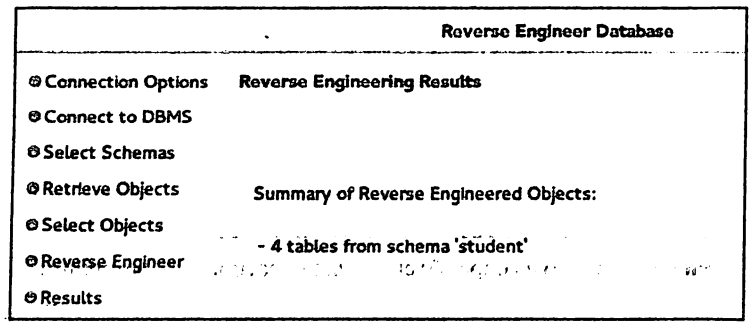


Рис. 5.49. Окно результатов обратного инжиниринга

ринга в физической схеме ER-модели появятся созданные таблицы (рис. 5.50).

Далее эти таблицы можно поместить на ER-диаграмму (рис. 5.51). По умолчанию генерация производится в той нотации, которая была настроена. Если нотация не была выбрана, то модель будет построена в нотации **Workbench Default**. Если необходимо получить модель в нотации **IDEF1X**, ее можно преобразовать при помощи вызова соответствующего пункта меню, как было показано ранее.

При необходимости полученную ER-модель можно дорабатывать, изменять, удалять, добавлять таблицы, менять поля и т. п. Однако следует заметить, что валидация моделей доступна только в коммерческой версии MySQL.

Если существует необходимость сгенерировать ER-модель из SQL-скрипта, то при выборе соответствующего пункта меню на глав-

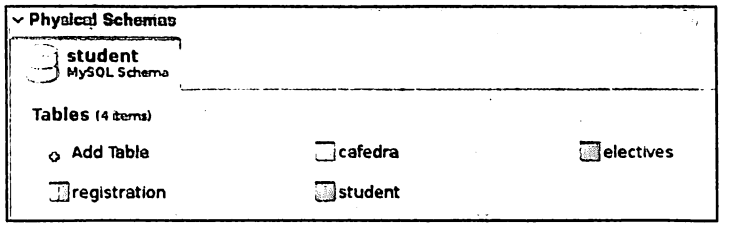


Рис. 5.50. Таблицы ER-модели, полученные при помощи обратного инжиниринга

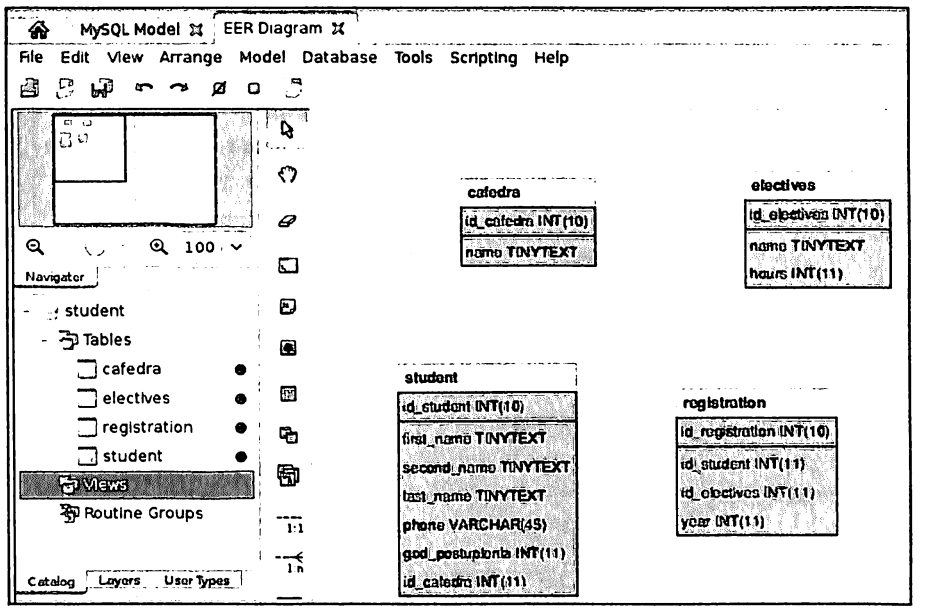


Рис. 5.51. Таблицы, полученные при помощи обратного инжиниринга на ER-диаграмме

ной странице **Create EER Model From SQL Script** (рис. 5.52) открывается окно (рис. 5.53). В этом окне через браузер можно выбрать файл, в котором хранится SQL-скрипт. Далее процесс создания ER-модели аналогичен описанному выше.

Последнее замечание касается назначения прав пользователя для моделей. В лабораторной работе 3 подробно был описан процесс создания нового пользователя и назначение ему прав для работы с конкретной базой данных. При создании модели добавить нового (заранее созданного) пользователя можно следующим образом. Перейти

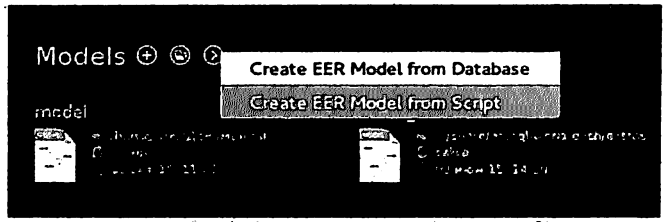


Рис. 5.52. Выбор пункта меню для генерации ER-модели из SQL-скрипта

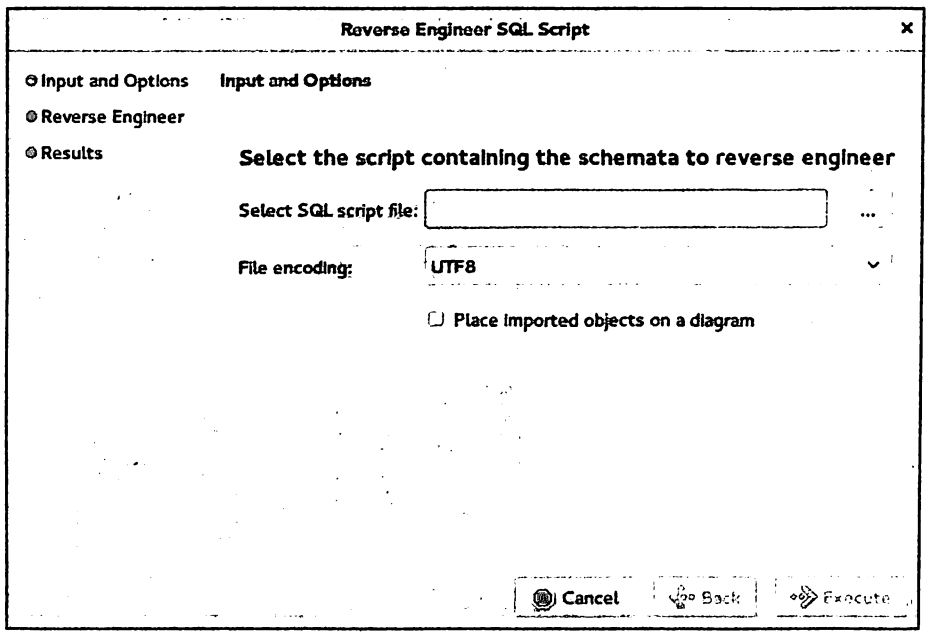


Рис. 5.53. Выбор файла для генерации ER-модели из SQL-скрипта

на вкладку **Shema Privileges** и нажать на кнопку **Add User**. Выбрать пользователя для добавления, например **user1**. После этого выбранный пользователь появится в соответствующей вкладке (рис. 5.54).

После этого пользователю можно назначить роль (рис. 5.55). Это может быть владелец, право чтения таблиц, вставки, изменения, выполнение скриптов или некоторая иная заранее определенная роль.

Рассмотренные выше принципы построения ER-моделей, прямой и обратный инжиниринг, позволяют в значительной мере упростить процесс проектирования баз данных и сделать его более наглядным.

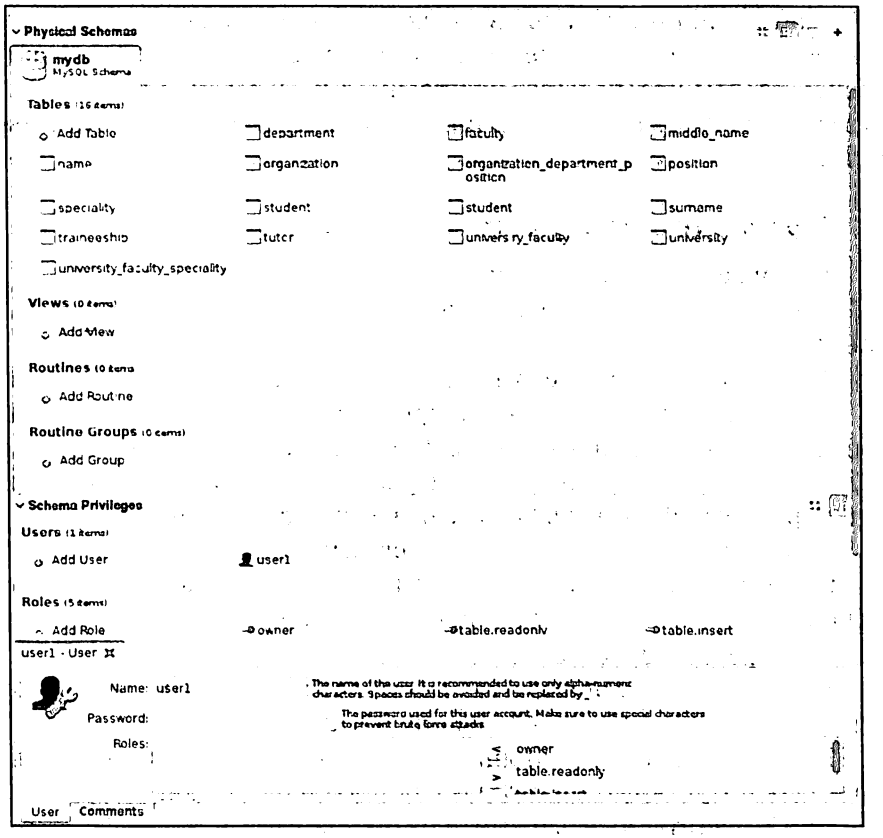


Рис. 5.54. Добавление нового пользователя

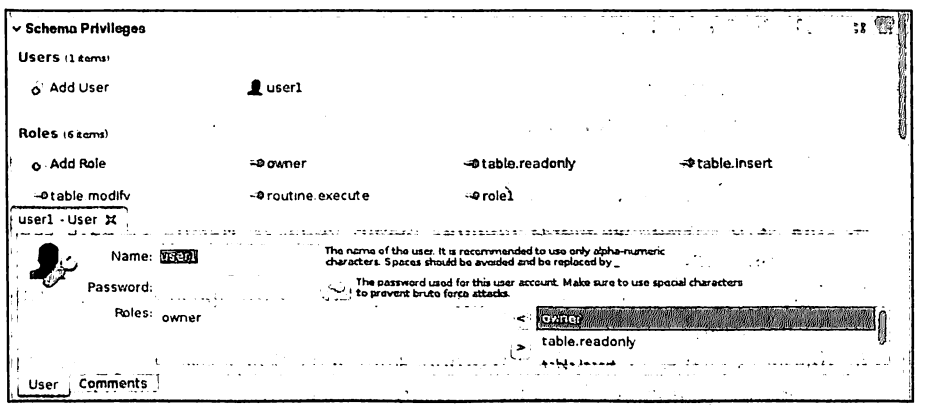


Рис. 5.55. Определение роли пользователя

Учащимся предлагается выполнить практическую работу, основанную на изученном теоретическом материале. Выполнение практической работы предполагает построение ER-моделей, а также осуществление прямого и обратного инжиниринга для выбранной предметной области.

Порядок выполнения практической работы

1. Ознакомиться с описанием.
2. Получить у преподавателя задание на проектирование базы данных.
3. Выделить основные сущности и атрибуты предметной области.
4. Открыть рабочую область программы MySQL Workbench для создания ER-моделей. Создать ER-модель по заданной предметной области. В каждой таблице задать первичный ключ и столбцы — атрибуты предметной области. Определить типы данных столбцов.
5. Определить типы и мощности связей между сущностями.
6. Создать связи между таблицами. Проверить наличие и расположение внешних ключей в дочерних таблицах.
7. Сохранить модель.
8. Получить у преподавателя название базы данных, для которой будет выполняться обратный инжиниринг.
9. Выполнить обратный инжиниринг для указанной базы данных. Сравнить полученную ER-модель с исходной базой данных.
10. Сохранить модель и выйти из программы.

Контрольные вопросы

1. Для каких целей используется построение ER-диаграмм?
2. Из каких шагов состоит процесс построения ER-модели таблицы?
3. Какие вкладки предназначены для создания столбцов таблицы? Внешних ключей? Ввода информации?
4. Для чего служат шаблоны таблиц? Каким образом можно создать свой шаблон таблицы?
5. Какая кодировка используется для корректного отображения содержимого таблиц?
6. Каким образом внешний ключ отображается на диаграмме в случае независимых сущностей? В случае зависимых сущностей? В случае независимых сущностей, допускающих значение **NULL**?

7. Что такое идентифицирующая связь? Неидентифицирующая связь? Каким образом они отображаются на диаграммах в рамках нотации **IDEF1X**?
8. Как отображается на диаграмме связь независимых сущностей, допускающих значение **NULL**?
9. Каким образом на диаграмме отображается мощность связи?
10. Какие настройки можно произвести во вкладке **Foreign Key**?
11. Из каких шагов состоит процесс прямого инжиниринга?
12. Для каких целей используется кнопка **Show Logs**?
13. Для каких целей используется процесс обратного инжиниринга? Опишите этапы обратного инжиниринга.
14. Можно ли сгенерировать ER-модель из скрипта? Каким образом?

Часть II
ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ СУБД NOSQL-ТИПА НА ПРИМЕРЕ MONGODB

Введение. Общие сведения о СУБД MongoDB

В последние годы появилось значительное количество систем управления базами данных (СУБД) СПО, которые обладают высокой производительностью, бесплатны и хорошо документированы. Кроме того, многие из них предназначены для использования в ОС Linux с широко распространенным сервером Apache, и многие языки программирования (PHP, Python, Ruby) имеют необходимые драйверы и функции, позволяющие достаточно просто наладить работу с этими СУБД.

Заметим также, что не все эти СУБД поддерживают реляционную модель данных: все больше начали распространяться базы данных, основанные на модели **NoSQL** («не только SQL»), что позволяет оптимальным образом хранить данные нерегулярной структуры, не создавая избыточности. Чаще всего такие СУБД имеют клиент-серверную архитектуру и являются распределенными. Иными словами, подавляющее большинство пользователей имеют интерфейс «тонкий клиент», поэтому основная часть задач по обработке информации переносится на некоторый ресурс, например сервер или облако, что также позволяет оптимизировать хранение данных и скорость вычислений.

MongoDB (от англ. *humongous* — огромный) является документно-ориентированной кросс-платформенной системой СУБД и относится к базам данных NoSQL. Написана на языке C++. MongoDB выпускается на условиях комбинации лицензий GNU Affero General Public License и Apache License и является бесплатной СУБД с открытым исходным кодом. Скачать версию для установки и документацию можно с сайта **http://www.mongodb.org** [6].

На сайте для MongoDB имеются пакеты дистрибутивов для различных ОС (Windows, Linux, MacOS) и различных версий (32- и 64-битные), однако ниже рассмотрены установка и выполнение примеров для 64-битной ОС Fedora 22, которая, как было сказано ранее, проста в установке, нетребовательна к ресурсам и является СПО.

Согласно последним данным MongoDB является наиболее популярной NoSQL базой данных, которая строится не на основе таблиц реляционной структуры, а на **JSON**-подобных документах, хранящих данные в виде «ключ—значение» (документно-ориентированное хранение), а также дает возможности гибкой поддержки индекса, автоматического шардинга, большого количества запросов, встроенных репликаций, поддержки спецификации хранения GridFS для хранения и извлечения файлов, превышающих предельный размер BSON-документа 16 МБ, и многие другие.

JSON (от англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком, считается независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

В MongoDB используется формат BSON (от англ. Binary JavaScript Object Notation) [5]. Это компьютерный формат обмена данными, используемый в основном в качестве формата для хранилища данных и передачи по сети в базе MongoDB. Это бинарная форма представления простых структур данных и ассоциативных массивов (так называемые объекты, или документы, в MongoDB). Название BSON основано на определении JSON и значит «Binary JSON» (бинарный JSON). Формат является надмножеством JSON, поскольку включает в себя дополнительно такие типы данных, как Date type и BinData type. Заметим, что кодирование данных в BSON и декодирование из BSON могут быть выполнены очень быстро в большинстве языков благодаря использованию типов данных Си.

Отличие форматов заключается в следующем: JSON-текст представляет собой (в закодированном виде) одну из двух структур.

1. Набор пар ***ключ:значение***. В различных языках это может быть реализовано как запись, объект, словарь, хэш-таблица, ассоциативный массив и т. п. Ключом может быть только строка. В качестве зна-

чений в JSON используются: объекты, числа, строки (под которыми естественным образом понимается упорядоченное множество из нуля или более символов юникода, заключенное в двойные кавычки, с использованием *escape*-последовательностей, начинающихся с обратной косой черты), ***true*** или ***false***, массивы, ***null***, вложенные структуры.

2. Упорядоченный набор ***значений***. Это может быть реализовано как массив, вектор, список или последовательность.

Современные языки программирования, как правило, поддерживают такие универсальные структуры данных в той или иной форме.

BSON-документы (объекты) состоят из сортированных списков элементов. Все документы в MongoDB хранятся в формате BSON. Каждый элемент состоит из имени поля, типа и значения. Именами полей являются строки.

```
{
field1: value1,
field2: value2,
field3: value3,
...
fieldN: valueN
}
```

Таким образом, отличие от реляционных баз данных MongoDB является документно-ориентированной и, в отличие от реляционных баз данных, не имеет четко определенного количества атрибутов сущности, что соответствует заранее определенному числу столбцов с заданным типом данных в реляционной СУБД.

Максимальный размер **документа BSON** — 16 Мб. Для хранения документов большего размера используется GridFS API — спецификация, описывающая дробление и сохранение больших файлов в MongoDB. MongoDB поддерживает не более 100 уровней вложенности для BSON-документов. GridFS состоит из двух коллекций. В первой коллекции (files) хранятся имена файлов, а также их метаданные, например размер. В другой коллекции (chunks) в виде небольших сегментов хранятся данные файлов.

На сегодняшний день спецификация BSON поддерживает 20 типов данных. Описание спецификации BSON в псевдо-БНФ (форма Бэкуса—Наура) приведено в приложении С (более подробно см.: URL: **http://bsonspec.org/#/specification**).

В MongoDB поддерживаются как стандартные типы данных, так и специфичные для MongoDB. Основные типы данных перечислены

ниже (полный список см.: URL: **http://docs.mongodb.org/manual/reference/ bson-types**):

- String — строка в кодировке UTF8;
- 32-bit и 64-bit integer — целое число;
- Floating point — число с плавающей точкой;
- Double — число с плавающей точкой двойной точности;
- Date — дата;
- Boolean — булевые (принимает два значения — «истина» и «ложь»);
- Regular Expression — регулярные выражения;
- Timestamp — специальный тип данных в MongoDB, используемый для репликации и шардинга;
- Null — «ничего» (неопределенное значение);
- Object — BSON-объект;
- ObjectId — специальный 12-байтный тип BSON, гарантирующий уникальность внутри коллекции; в MongoDB используется как значение по умолчанию для поля _id;
- Symbol — символ (для языков с иными символьными типами);
- JavaScript — код JavaScript.

Формально множество типов BSON является расширением JSON-типов, поскольку в JSON нет, например, типа массива байт. Однако в BSON существуют ограничения по длине. Кроме того, некоторые значения, допустимые в JSON, не допускаются в BSON (см. выше).

Заметим, что имена, в том числе базы данных коллекции, должны быть короче, чем 123 байта. По умолчанию размер файла имен 16 МБ. Он может быть настроен при помощи опции **nssize**. Изменить размер можно, например, в конфигурационном файле **mongodb.conf**. Поиск файла осуществляется командой

```
$ whereis mongodb
```

Ответ — местоположение файла:

```
mongodb: /etc/mongodb.conf
```

Далее найдите соответствующую опцию, снимите комментарии и произведите настройки по общему принципу **<setting> = <value>**, в нашем случае **nssize = <size>**.

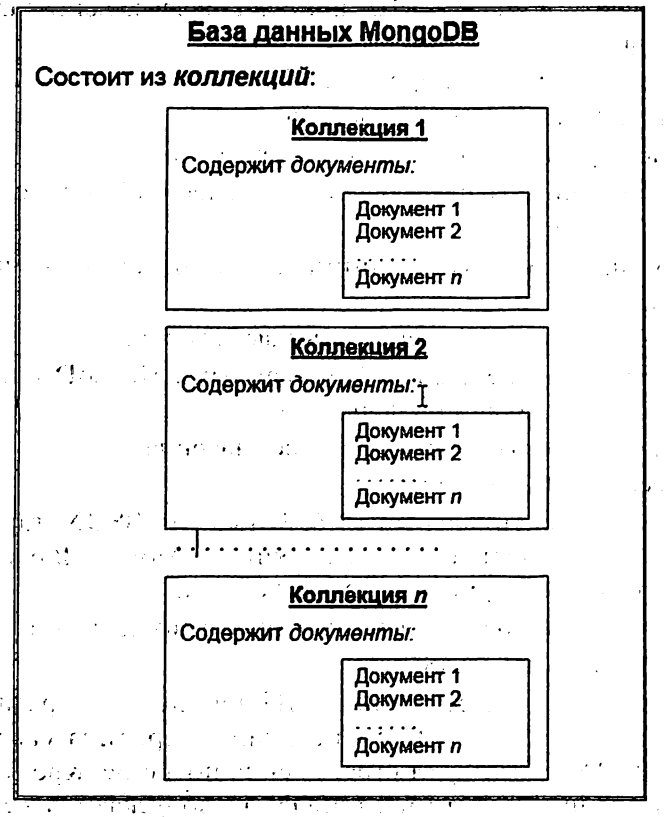


Рис. Структура базы данных MongoDB

Внимание! Имена баз данных чувствительны к регистру. Однако MongoDB не позволяет создавать базы данных, которые различаются только регистром букв и (или) содержат в названии спецсимволы.

Общий вид базы данных MongoDB представлен на рисунке.

Таким образом, очевидны преимущества MongoDB для хранения разнородной документно-ориентированной информации.

Лабораторная работа 6
НАЧАЛО РАБОТЫ С MONGODB: УСТАНОВКА СУБД И ОБОЛОЧКИ ROBOMONGO И СОЗДАНИЕ ТЕСТОВОЙ БАЗЫ ДАННЫХ

Цель: изучение принципов установки и работы с СУБД MongoDB.

Установка СУБД и оболочки

Для установки mongo в терминале в командной строке следует набрать команду:

```
# yum install mongodb mongodb-server mongodb-devel
php-pecl-mongo
```

Далее ответить на вопрос:

```
this is ok [y/d/N] y
```

После этого необходимо создать директорию, в которой **mongo** будет искать данные **data/db** (**mongo** всегда сперва обращается к директории **data/db**, если же есть желание хранить в другой директории — необходимо конфигурировать **mongo**). В корне создается директория **data** и поддиректория **db**:

```
# mkdir -p /data/db
```

Теперь следует установить владельца директории:

```
# chown mongodb /data/db
```

Запустим mongoDB:

```
# service mongod start
```

Перезапустим **Apache**:

```
# service httpd restart
```

И далее следует команда перезапускать MongoDB при старте:

```
# systemctl start mongod.service
```

Настройка **SELinux**:

```
# yum install policycoreutils-devel # this is to install
audit2app
# grep mongod /var/log/audit/audit.log | audit2allow -M mypol
# semodule -i mypol.pp
# mkdir -p /var/lib/mongo
# semanage fcontext -a -t mongod_tmp_t
'/var/lib/mongo/mongod.lock'
```

Всегда на всякий случай при старте (перезагрузке) следует выполнять команду:

```
# service httpd restart
```

Для ввода тех или иных команд надо набрать в командной строке **mongo**, т. е.

```
$ mongo
```

После этого получаем приглашение:

```
MongoDB shell version: 3.0.8
connecting to: test
>
```

Затем можно набирать команды в командной строке для работы с базой данных MongoDB.

Для выхода из оболочки **mongo** необходимо набрать команду

```
> exit
```

Создание новой базы данных

В первую очередь еще раз обратим внимание на различия в терминологии между реляционными базами данных и MongoDB, приведенные в табл. 6.1: так, термину **таблица** в реляционной базе данных соответствует термин **коллекция** и т. п.

При создании новой базы данных помните, что MongoDB во многом схожа с привычной концепцией реляционных баз данных. Базы

Таблица 6.1. **Различие терминов и понятий**

SQLs	MongoDB
Таблица	Коллекция
Строка	Документ или BSON-документ
Объединение таблиц	Встроенные документы и ссылки
Первичный ключ (в качестве первичного ключа выступает одно или несколько полей (столбцов), комбинация значений которых однозначно определяет каждую запись в таблице	Первичный ключ (автоматически устанавливается в поле **_id**)

данных хранятся на сервере MongoDB. База данных содержит коллекции, являющиеся (см. табл. 6.1) аналогом таблиц и состоящие из документов, которые, в свою очередь (см. табл. 6.1), являются аналогом строк реляционной базы данных и содержат минимум одно поле.

Для начала работы создадим новую базу данных **staff**. Для сравнения процесса будем совершать аналогичные операции с реляционной базой данных (в данном случае MySQL в среде Workbench). Заметим, что, возможно, через некоторое время MySQL будет замещаться на MariaDB (а в некоторых версиях Linux замена уже произведена), что связано с политикой лицензирования MySQL. Однако разработчику это не доставит неудобств, поскольку API, протоколы, библиотеки и приложения MariaDB и MySQL совместимы.

Для создания базы данных в MySQL в среде Workbench [1] во вкладке **Overview** выберите вторую слева пиктограмму **Create a New Schema**. Задайте имя базы данных, кодировку таблицы. Далее необходимо каждый раз для подтверждения нажимать кнопку **Apply** внизу страницы, после чего можно приступать к созданию таблиц. Для этого выберите третью слева пиктограмму с названием **Create a New Table**. После создания таблиц и их заполнения база данных готова к работе.

Для работы с MongoDB необходимо запустить (см. выше) сервер **mongod** (запуск сервера зависит от используемой сборки Linux) и консоль **mongo** (в командной строке терминала наберите **mongo**).

Получаем ответ и знак «>» — приглашение к работе:

```
MongoDB shell version: 2.4.13
connecting to: test
>
```

Для проверки баз данных, доступных на сервере, можно использовать команду

```
> show dbs
```

В качестве ответа будут выданы все базы данных, находящиеся на сервере, например, если помимо двух служебных баз данных (**admin** и **local**) имеется база данных **users**:

```
admin (empty)
local 0.078125GB
users 0.203125GB
>
```

Предположим, что на данный момент не имеется никаких пользовательских баз данных, только две служебные. Для создания базы данных в MongoDB в командной строке набираем:

```
> use test
```

В качестве ответа получаем:

```
switched to db test
```

Обратите внимание, что база данных создается в момент создания первой коллекции, т. е. если просто ввести команду **> use test**, база создана не будет:

```
> use test
switched to db test
> show dbs;
admin (empty)
local 0.078125GB
```

Коллекция создается командой **insert()**, например:

```
> db.collection1.insert({ name: "Соколов"})
```

Для удаления коллекции (подробнее см. ниже) следует использовать метод **drop()**.

Создадим коллекцию MongoDB. Для этого сначала необходимо проверить нахождение в нужной базе данных, в противном случае коллекция будет создана в той базе данных, в которой находится

пользователь. Проверить, какая база данных является текущей, можно командой

```
> db.stats()
```

Поскольку ранее была выполнена команда

```
> use test
```

то коллекция будет создана именно в этой базе.

Сделаем еще одно замечание, касающееся уникального идентификатора. Для каждого документа в MongoDB (как и для каждой записи в реляционной базе данных) должен быть определен уникальный идентификатор. Поле уникального идентификатора называется **_id**. Существует два способа задать уникальный идентификатор. Его значение может быть явно указано при заполнении данных. Если значение не указано явно, то при добавлении документа в коллекцию идентификатор создается автоматически. В этом случае генерируется специальное значение длиной 12 байт, состоящее из нескольких сегментов, гарантирующих уникальность объекта **ObjectId**.

Итак, создадим новую коллекцию. В командной строке наберите:

```
> db.test1.insert( {
... id_test1: "id1",
... name: "Иванов",
... } )
>
```

Для проверки содержимого коллекции (создана ли она) в командной строке наберите:

```
> db.test1.find()
```

В ответ получаем содержимое коллекции:

```
{"_id": ObjectId("530db671aeef2e9d5ba4188c"), id_test1: "id1",
name: "Иванов",}
>
```

Обратите внимание, что в запросе используется функция **find()**, которая является некоторым аналогом хорошо известной функции **SELECT** реляционных баз данных.

Обратите внимание, что в ответе на запрос явно появилось дополнительное поле **_id**, поскольку, как было сказано выше, каждый

документ должен иметь уникальный идентификатор. Ниже будет показано, каким образом оно может быть сгенерировано самим пользователем.

В предыдущем случае коллекция была создана неявным образом, путем вставки в нее некоторых данных. Для явного создания коллекции следует использовать команду:

```
>db.createCollection("имя_коллекции")
```

Если необходимо переименовать коллекцию, то команда имеет следующий вид:

```
>db.test1.renameCollection("новое_название")
```

Удалим созданную коллекцию **test1** в базе данных **test** и далее рассмотрим, каким образом можно создать базу данных и коллекцию в графической оболочке Robomongo.

Внимание! Если создать вместо удаленной базы данных и/или коллекции новую базу данных и/или коллекцию с теми же названиями базы данных, коллекции и полей, то при внесении данных автоматически генерируемое значение поля **ObjectId** изменится, поскольку для его формирования используется сложный алгоритм вычисления, включающий в качестве одной из величин системное время. Из сказанного следует невозможность получить совпадающие значения поля **ObjectId** для различных документов.

Установка и начало работы с оболочкой Robomongo

Для удобства работы можно использовать некий вариант графической оболочки, которых разработано достаточно много. Однако наиболее удобной является оболочка **Robomongo** (СПО). Robomongo — кроссплатформенная (Windows, OS X, Linux) графическая оболочка для работы с СУБД MongoDB. Robomongo является программным обеспечением с открытым исходным кодом, для работы с которой используется тот же движок, что и для JavaScript на основе Mozilla SpiderMonkey.

Robomongo позволяет работать с большим удобством, поскольку имеет такие возможности, как подсветка, автодополнение, различные режимы просмотра (текст, дерево, пользовательские) и многое другое. Также удобно иметь возможность открыть несколько оболочек

как для одной базы данных (они будут изолированы друг от друга), так и для различных баз данных.

Для установки необходимо зайти на сайт **http://robomongo.org** (рис. 6.1) и на сайте выбрать соответствующую операционную систему (рис. 6.2).

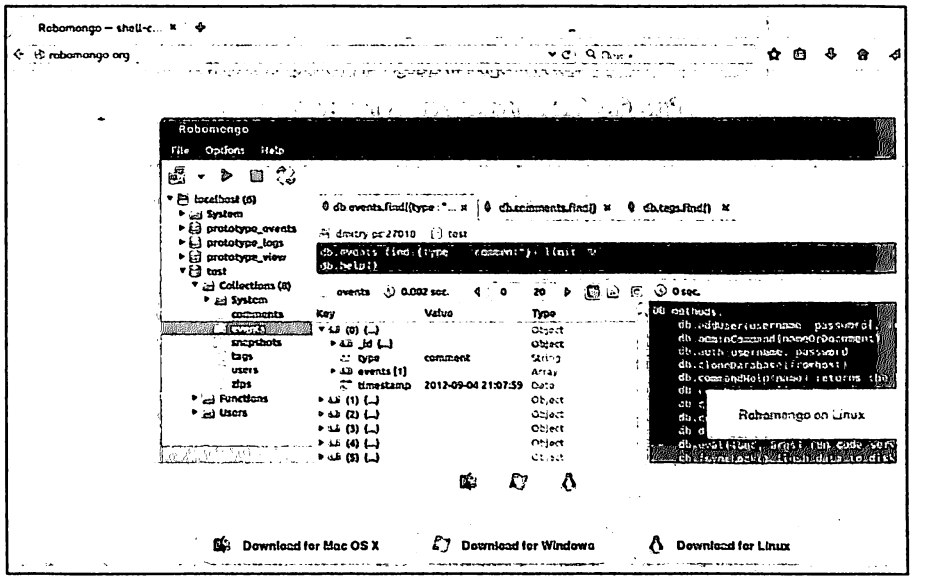


Рис. 6.1. Сайт Robomongo

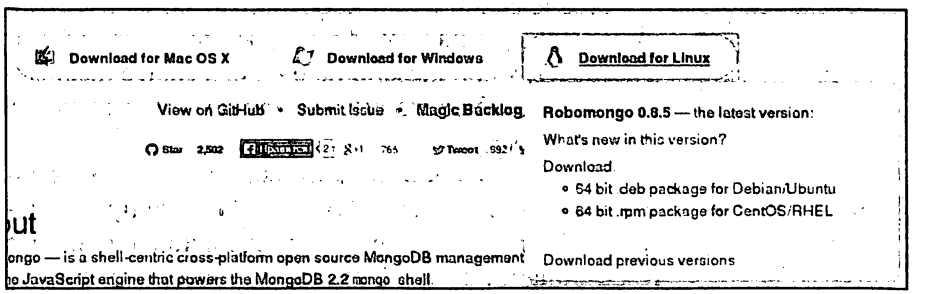


Рис. 6.2. Выбор ОС для скачивания Robomongo

Поскольку вся разработка происходит на ОС Linux Fedora, соответственно выбирается загрузочный файл для ОС Linux и rpm пакет. Данный пакет сохраняем (рис. 6.3).

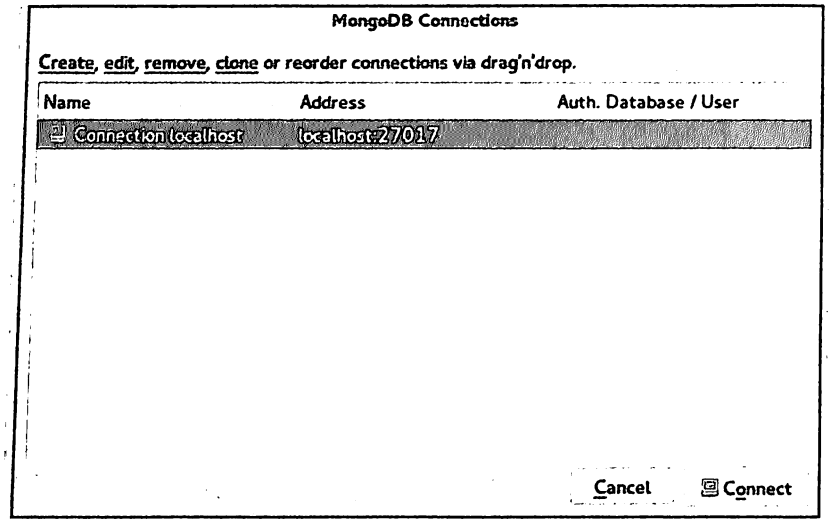


Рис. 6.8. Соединение с сервером установлено

чами). После настройки соединение появляется на экране (рис. 6.8). Соединение можно редактировать, удалять, дублировать.

После щелчка на соединении открывается экран, как показано на рис. 6.9. В соединениях указано три базы данных, хотя отображена только одна под названием «test», остальные две являются служебными (об этом было сказано ранее), и в соединении они не показаны.

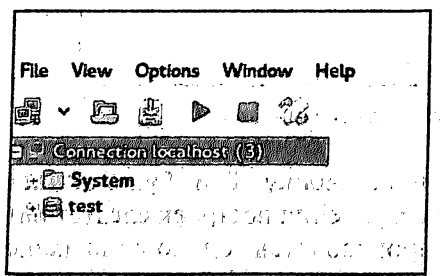


Рис. 6.9. Вид оболочки Robomongo

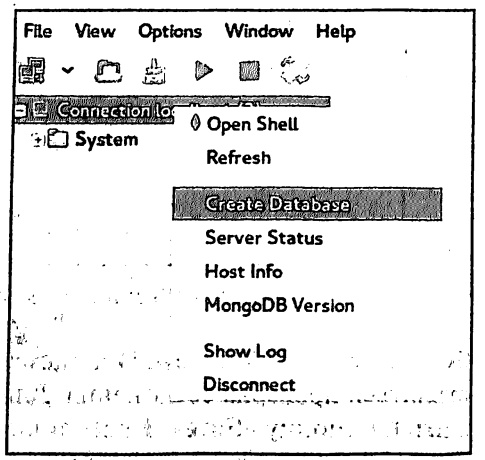


Рис. 6.10. Создание новой базы данных

Далее для простоты вся работа с базой данных будет производиться через оболочку Robomongo (однако если работать через командную строку, то результат будет аналогичен, хотя и менее нагляден).

Для создания новой базы данных необходимо щелкнуть правой клавишей мыши на соединении и выбрать пункт меню «Create Database», как показано на рис. 6.10. И в появившемся окне (рис. 6.11) ввести название базы, например **test**.

После нажатия на кнопку «Create» будет создана база данных (рис. 6.12).

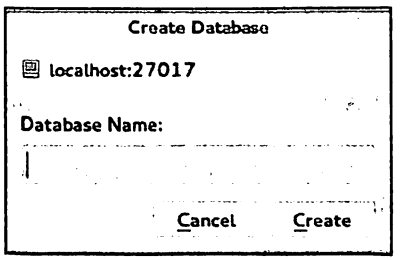


Рис. 6.11. Окно для внесения названия новой базы данных

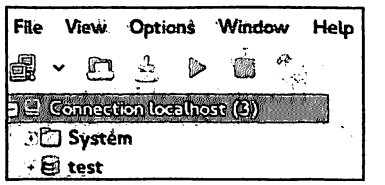


Рис. 6.12. Создана новая база данных

Если щелкнуть мышью на знак «+», то в раскрытом списке будет видно, что в базе данных **test** имеется возможность создавать коллекции, функции и пользователей (рис. 6.13). Заметим, что число коллекций при создании базы данных не равно нулю, поскольку имеется коллекция со служебной информацией.

Для создания новой коллекции переходим в списке на коллекции («**Collections**») и выбираем пункт «Create Collections» (рис. 6.14) и вно-

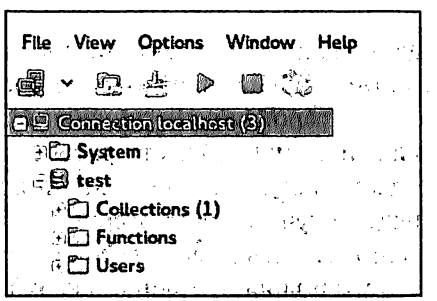


Рис. 6.13. Коллекции, функции, пользователи

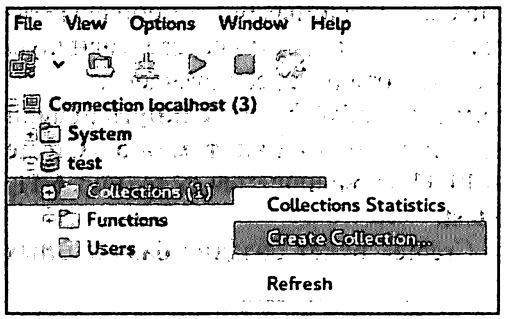


Рис. 6.14. Создание новой коллекции

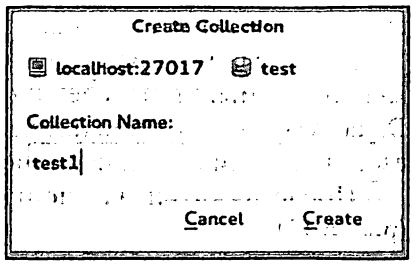


Рис. 6.15. Внесение названия новой коллекции

сим название коллекции (например, **test1**) в открывшемся окне (рис. 6.15).

После этого в пункте **«Collections»** изменится количество коллекций, т. е. добавится созданная (рис. 6.16).

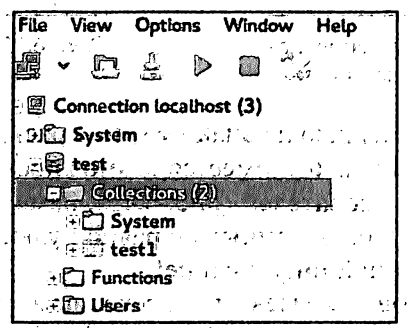


Рис. 6.16. Изменение числа коллекций

Дважды щелкнув мышью на коллекции, попадаем в рабочее пространство оболочки. Верхняя часть рабочего пространства является командной строкой, в которой можно вносить данные в коллекции, составлять запросы и т. п. В нижней части рабочего пространства оболочки будут представлены результаты выполнения команд (рис. 6.17).

Как и в предыдущем случае, для проверки баз данных, доступных на сервере, можно использовать команду **show dbs** (рис. 6.18).

Рассмотрим немного подробнее меню в верхней части окна оболочки. Пункт меню **«File»** предназначен для открытия и сохранения файлов (рис. 6.19).

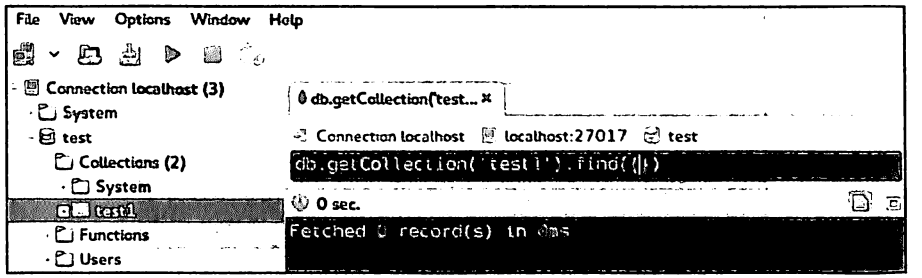


Рис. 6.17. Рабочее пространство оболочки

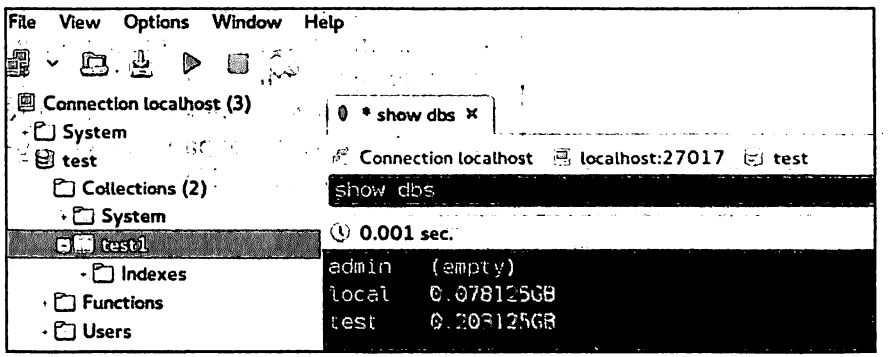


Рис. 6.18. Проверка баз данных, имеющихся на сервере

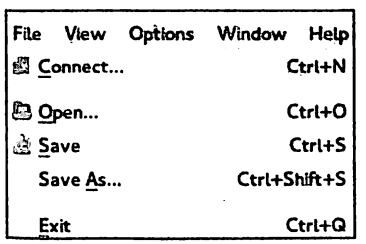


Рис. 6.19. Пункт меню «File»

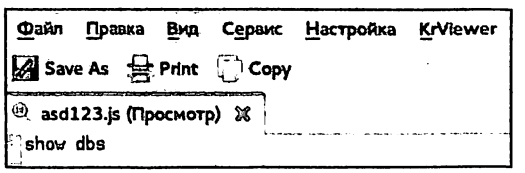


Рис. 6.20. Сохранение файла

Если код, содержащийся в командной строке, необходимо сохранить в файле, то выбирается соответствующий пункт меню (**File Save** или **Save as**), и код будет сохранен в файле с расширением **js** (рис. 6.20). Для запуска сохраненного файла его необходимо открыть, выбрав пункт меню **File → Open** (рис. 6.21) и нажать кнопку  (подробнее она будет описана ниже).

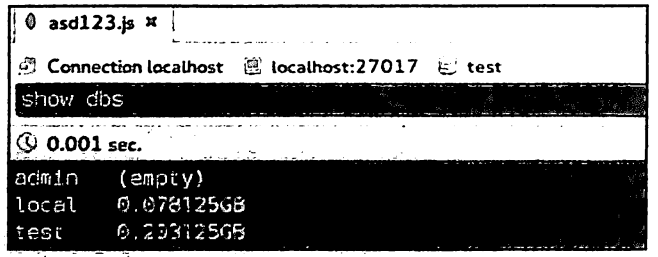


Рис. 6.21. Загрузка файла

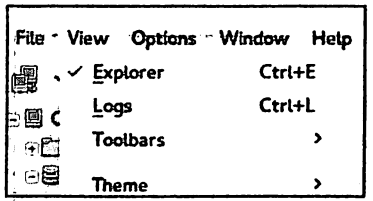


Рис. 6.22. Пункт меню «View»

Пункт меню «**View**» предназначен для настройки вида экрана, например отображения на экране логов и панелей инструментов (рис. 6.22). Так, если поставить галочку на пункте меню «Log», то на экране будут отображены логи (рис. 6.23).

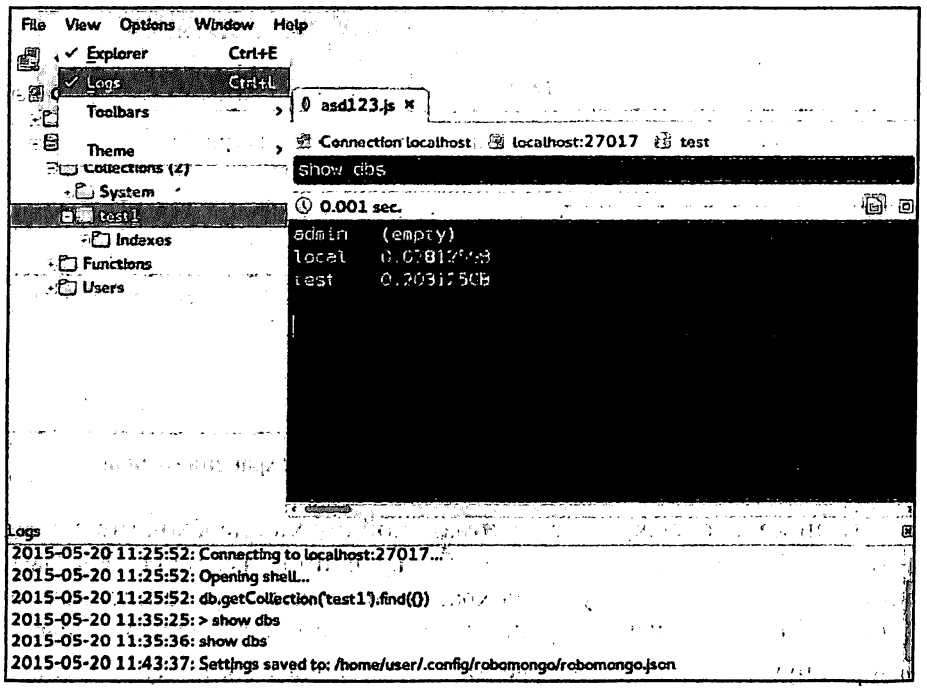


Рис. 6.23. Отображение логов

Пункт меню «**Options**» предназначен для дополнительных настроек (рис. 6.24). Например, можно настроить отображение времени в UTC или задать местное (рис. 6.25) и т. п.

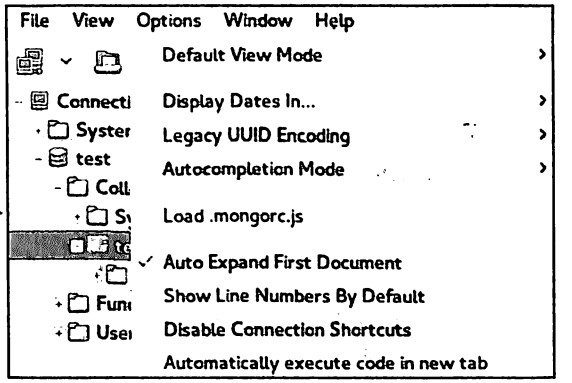


Рис. 6.24. Пункт меню «Options»

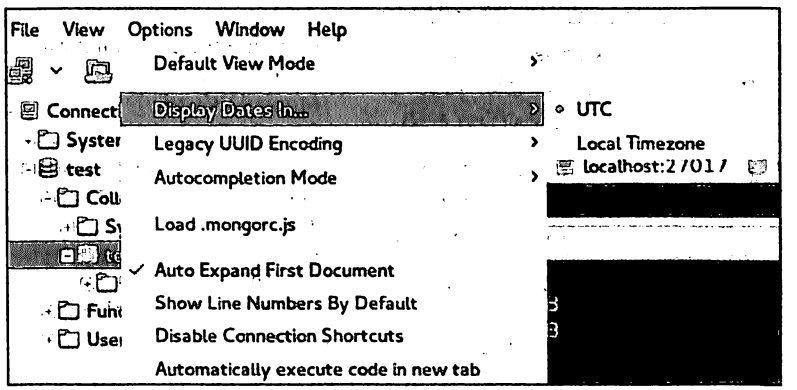


Рис. 6.25. Задание отображения времени

Пункт меню «**Windows**» предназначен для задания параметров окна (рис. 6.26).

Пункт меню «**Help**» является традиционным и содержит информацию об оболочке Robomongo.

Ниже расположены пиктограммы панели управления (рис. 6.27). Их значения и горячие клавиши для вызова приведены в табл. 6.2.

Вернемся к созданию коллекции **test1** в тестовой базе данных **test**. Как было сказано выше, для вставки записей используется команда **insert**. Создадим следующие поля в коллекции: идентификатор и имя

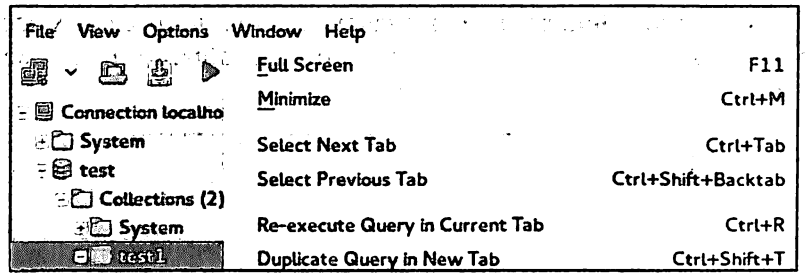


Рис. 6.26. Пункт меню «Windows»

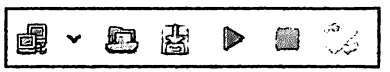


Рис. 6.27. Пиктограммы панели управления

Таблица 6.2. **Назначения пиктограмм и горячие клавиши панели управления**

Пиктограмма	Всплывающая подсказка	Значение
	**Connect to local or remote MongoDB instance (Ctrl + N)**	Соединение с локальным или удаленным сервером MongoDB(Ctrl+N)
	**Load script from the file to the currently opened shell (Ctrl + O)**	Загрузка скрипта из файла в открытую в данный момент оболочку (Ctrl+O)
	**Save script of the currently opened shell to the file (Ctrl + S)**	Сохранить файл из открытой в данный момент оболочки (Ctrl+S)
	**Execute query for current tab. If you have some selection in query text - only selection will be executed (F5 or Ctrl + Enter)**	Выполнить запрос в текущей вкладке. Если есть запрос на выборку в тексте запроса, то будет выполнена только выборка (F5 или Ctrl+Enter)
	**Stop execution of currently running script. (F6)**	Остановить выполнение текущего скрипта (F6)
	**Toggle orientation of results view (F10)**	Переключение ориентации отображенных результатов (F10)

(рис. 6.28). После нажатия на кнопку ▶ данные будут занесены в базу, и сообщение об этом появится в нижней части окна оболочки (рис. 6.29).

Если для наглядности необходимо отобразить коллекцию в виде дерева, необходимо нажать крайнюю левую кнопку (рис. 6.30).

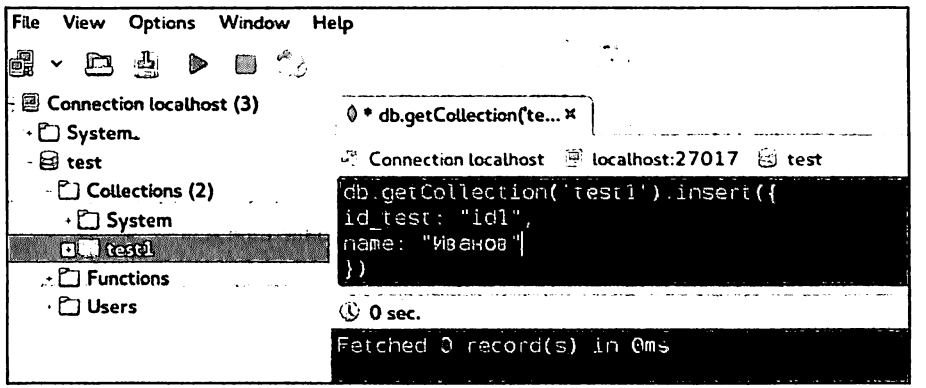


Рис. 6.28. Заполнение коллекции

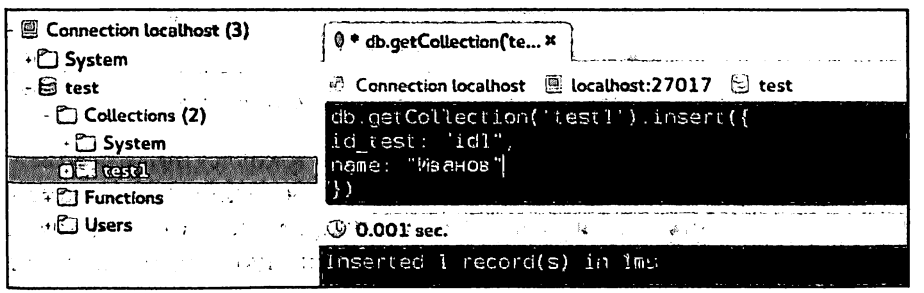


Рис. 6.29. Данные внесены

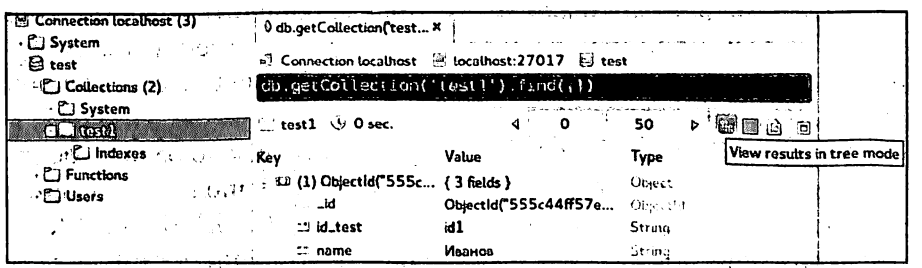


Рис. 6.30. Отображение коллекции в виде дерева

Если для наглядности необходимо отобразить коллекцию в виде таблицы, необходимо нажать на вторую слева кнопку (рис. 6.31).

Следующая кнопка позволяет отобразить коллекцию в текстовом виде (рис. 6.32).

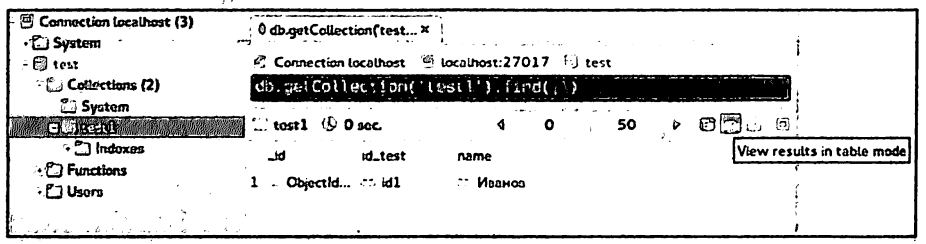


Рис. 6.31. Отображение коллекции в виде таблицы

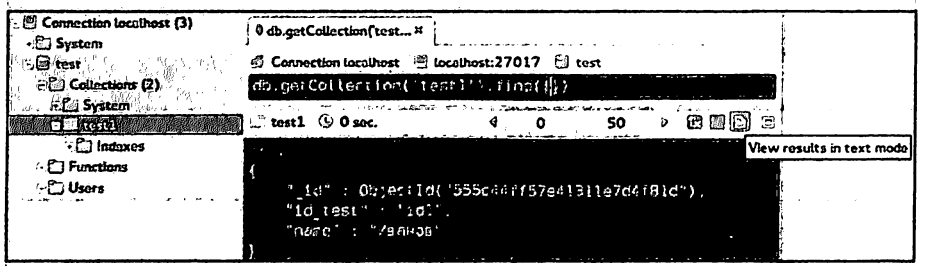


Рис. 6.32. Отображение коллекции в текстовом виде

Заметим, что после создания базы данных и коллекции в соответствующих пунктах меню становятся доступны различные команды для работы с ними. Для базы данных, например, доступны такие пункты меню, как обновление, просмотр статистики, восстановление и удаление (рис. 6.33).

Для коллекции доступны следующие пункты меню. При работе с документами данной коллекции: вставить, обновить, удалить. При работе непосредственно с самой коллекцией: переименовать, дублировать, удалить, получить статистическую информацию о коллекции (рис. 6.34) и т. п.

Учащимся предлагается выполнить практическую работу, основанную на изученном теоретическом материале. Выполнение практической работы предполагает изучение установки СУБД MongoDB, запуска графической оболочки Robomongo и создание тестовой базы данных.

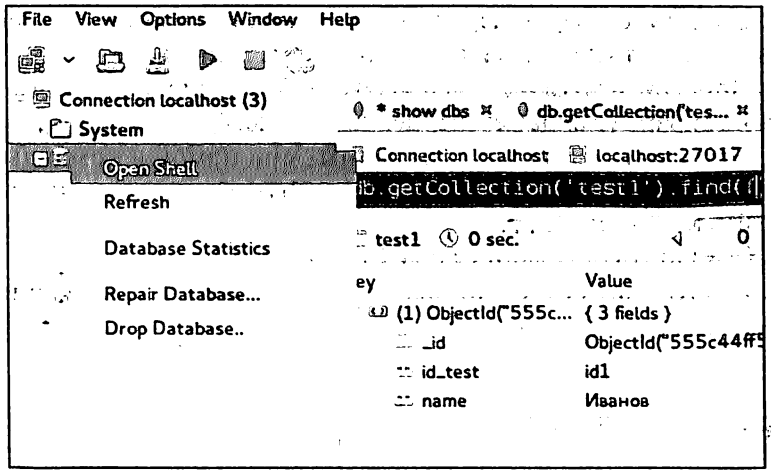


Рис. 6.33. Доступные операции для базы данных

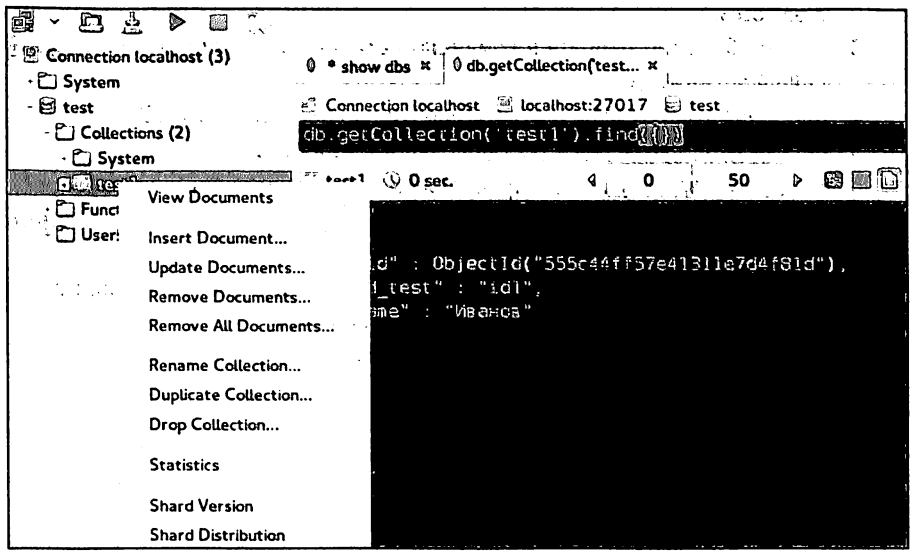


Рис. 6.34. Доступные операции для коллекции

Порядок выполнения практической работы

1. Ознакомиться с описанием.
2. Запустить оболочку **mongo**. Проверить наличие баз на сервере.
3. Запустить графическую оболочку **Robomongo**.

4. Создать тестовую базу данных из одной коллекции и одного документа, выбрать для полей различные типы данных.

5. Проверить возможные способы отображения созданного документа на экране (в виде дерева, таблицы, текста).

6. Проверить создание базы данных при помощи команды **show dbs**. Сохранить данную команду в файл. Вызвать команду из файла и запустить на выполнение.

7. Сохранить базу данных и выйти из программы.

8. Подготовить отчет по практической работе, содержащий основные этапы работы с тестовой базой данных.

Контрольные вопросы

1. В чем принципиальное отличие **NoSQL** базы данных от реляционной?
2. Перечислите основные типы данных, которые поддерживаются в **MongoDB**.
3. Что необходимо набрать в командной строке после установки **MongoDB** для ввода команд?
4. Какие вы знаете графические оболочки для работы с **MongoDB**?
5. Расскажите о возможностях оболочки **Robomongo**.
6. Какая команда служит для проверки баз данных, доступных на сервере?
7. Какую команду необходимо использовать для проверки содержимого коллекции?
8. База данных создается в момент создания первой коллекции или в момент ввода команды **> use "имя_базы_данных"**?
9. Какая команда служит для удаления коллекции?

Лабораторная работа 7
СОЗДАНИЕ, ОБНОВЛЕНИЕ И УДАЛЕНИЕ ДОКУМЕНТОВ В КОЛЛЕКЦИИ СУБД MONGODB

Цель: формирование навыков работы с документами коллекции созданной базы данных. Изучение возможностей оболочки Robomongo для работы с коллекциями.

Создадим базу данных под названием **staff**. Предположим, что первоначально необходимо создать одну коллекцию (таблицу) **personnel**, в которой будут храниться данные о сотрудниках:

- первичный ключ — уникальный идентификатор (целое число);
- идентификатор сотрудника (это может быть его табельный номер, пароль входа в систему и т. п.; поскольку он может содержать буквы и цифры, использование в качестве первичного ключа нежелательно) (строковый тип данных);
- фамилия сотрудника (строковый тип данных);
- телефон (строковый тип данных);
- возраст (целое).

Очевидно, что в реальных базах данных возраст задается иным способом, однако для учебного примера будем использовать упрощенную схему.

Для сравнения посмотрим (например, в [1]), как создается таблица в MySQL Workbench (рис. 7.1).

Код создания таблицы приведен ниже:

```
CREATE TABLE `staff`.` personnel ` (
 `id` INT UNSIGNED NOT NULL AUTO_INCREMENT ,
 `id_worker` VARCHAR(45) NOT NULL ,
 `name` TINYTEXT NULL ,
 `phone` TINYTEXT NULL ,
 `age` INT NULL ,
PRIMARY KEY (`id`) ,
UNIQUE INDEX `id_UNIQUE` (`id` ASC) ,
UNIQUE INDEX `id_worker_UNIQUE` (`id_worker` ASC) );
```

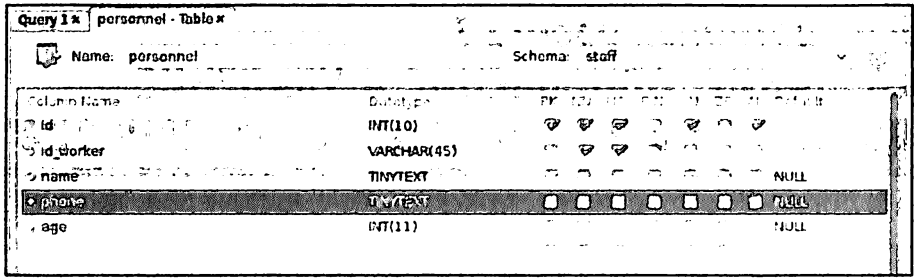


Рис. 7.1. Создание таблицы в MySQL Workbench

Как было сказано ранее, коллекция в MongoDB создается при помощи метода **insert()** — базового метода добавления информации в MongoDB. И если в документе не определено поле **_id**, то MongoDB добавит это поле и присвоит ему уникальный **ObjectId** документа, прежде чем вставить документ в коллекцию.

Для создания базы данных, как было показано выше, используем соответствующий пункт меню **«Create Database»** в **«Connections»** (см. выше) или команду **use**. Для создания баз данных **staff** команда будет иметь вид **use staff** (рис. 7.2). Обратите внимание, что в нижней части окна оболочки подтверждение переключения на базу данных **staff**:

```
switched to db staff
```

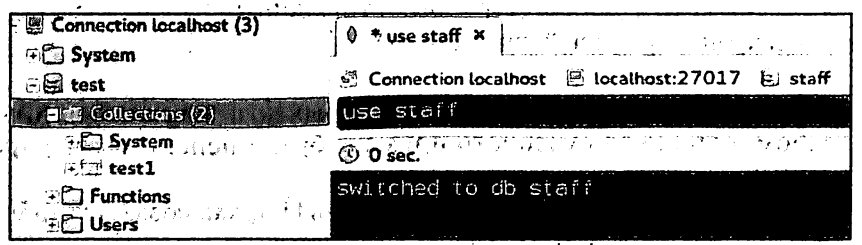


Рис. 7.2. Создание базы данных

Добавим нового сотрудника по фамилии Петров в нашу базу данных, определив поле **_id** (рис. 7.3):

```
db.personnel.insert( {_id: 123, id_worker: "w2",name:
"Петров",phone: "916 222 222 22",age: 65} );
```

Наличие точки с запятой в конце помогает MongoDB определять конец команды, хотя не является обязательным.

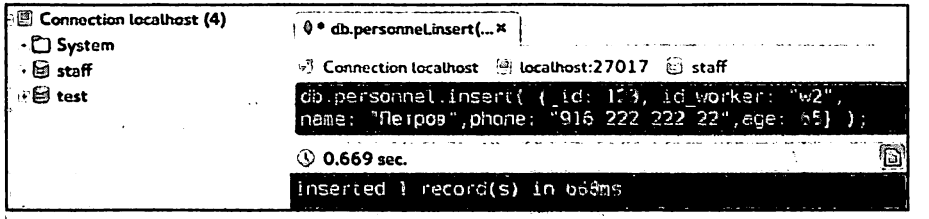


Рис. 7.3. Внесение данных в базу

Проверьте, что запись внесена (рис. 7.4). Чтобы свернуть запись, необходимо нажать на знак «—» слева от уникального идентификатора. Обратите внимание, что названия полей упорядочиваются по алфавиту.

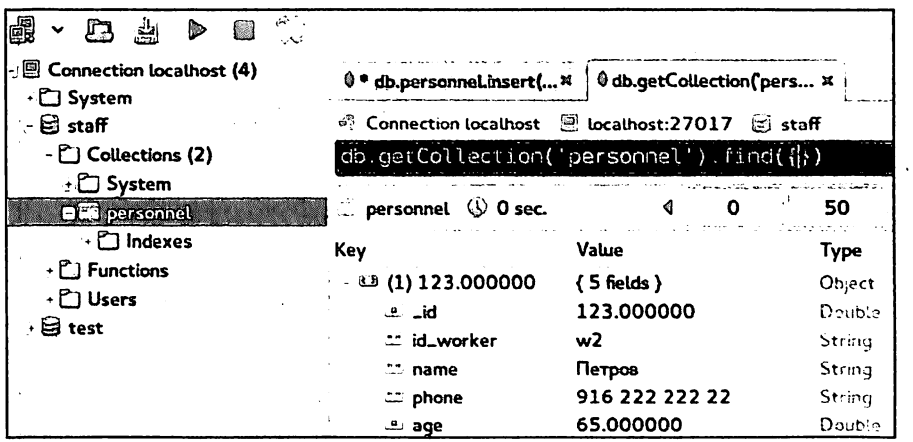


Рис. 7.4. Данные в базу внесены

Внесем данные о сотруднике Иванове (рис. 7.5). Поле идентификатора определять не будем:

```
db.personnel.insert(id_worker: "w1",name: "Иванов",phone: "916
111 11 11",age: 25} );
```

На рис. 7.6 показаны все имеющиеся на данный момент документы коллекции.

Обратите внимание, что во втором документе уникальный **ObjectId** был сгенерирован автоматически. Таким образом, уникальный **ObjectId** документа может быть присвоен полю **_id** автоматиче-

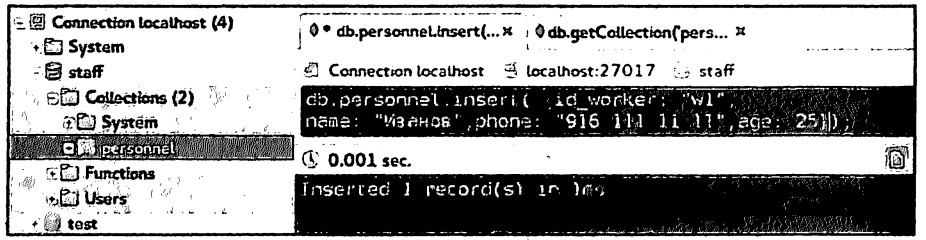


Рис. 7.5. Внесение документа без поля _id

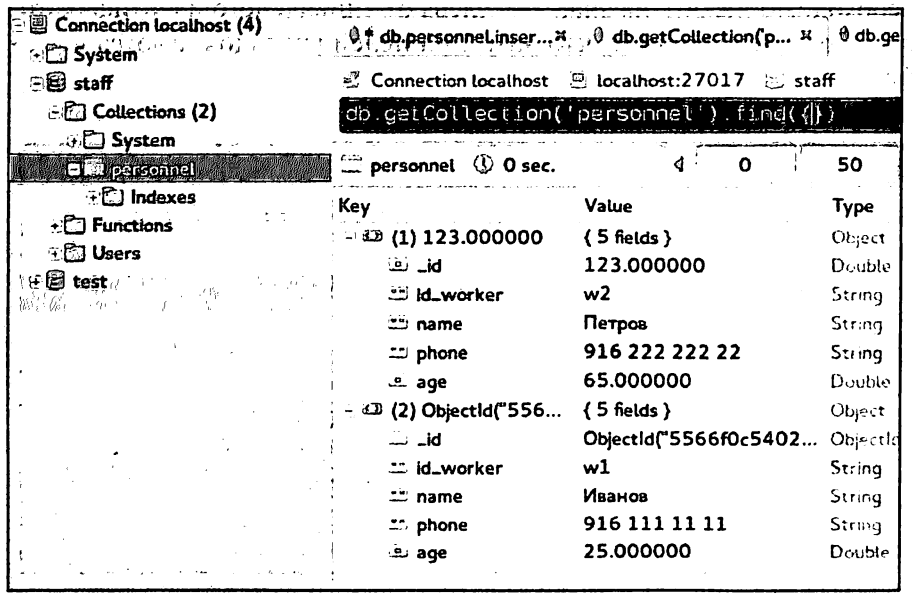


Рис. 7.6. Документы коллекции

ски или определен при создании документа. Покажем возможность вставки в коллекцию нескольких документов (рис. 7.7).

```
db.personnel.insert( [
{id_worker: "w3",name: "Сидоров",phone: "916 333 33 33",age:
45},
{ id_worker: "w4",name: "Смирнов",phone: "916 444 44 4",age:
35}
] );
```

Результат вставки (рис. 7.8).

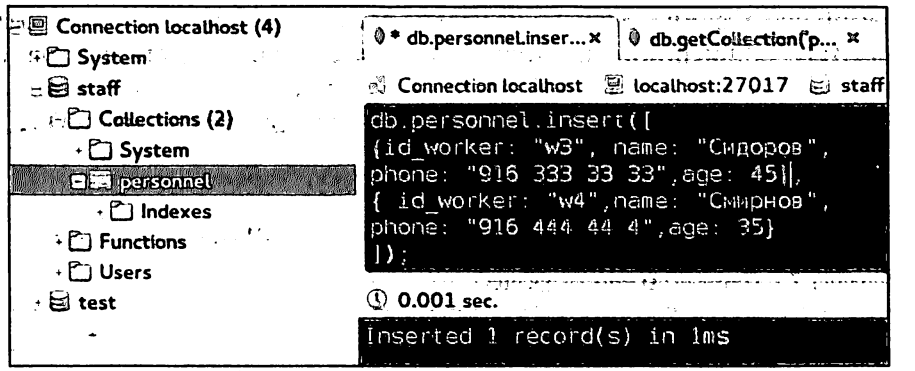


Рис. 7.7. Вставка нескольких документов в коллекцию

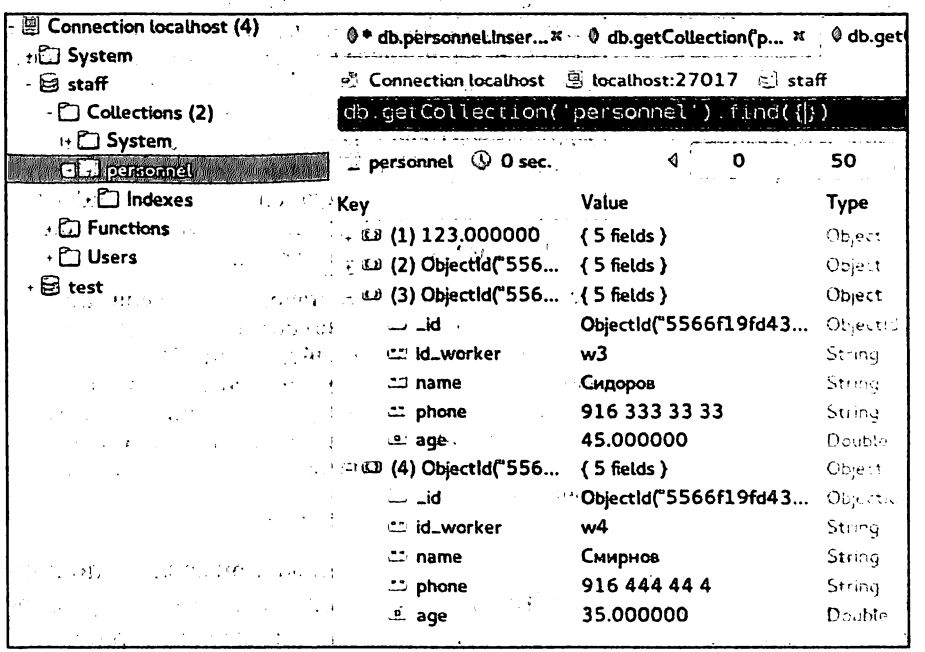


Рис. 7.8. Результат вставки нескольких документов в коллекцию

Рассмотрим, каким образом в MongoBD происходит **изменение коллекций** (например, добавление и удаление столбцов). В языке SQL этим изменениям соответствует команда **ALTER TABLE**.

Поскольку коллекции не описывают и не регламентируют структуру документа, то не существует такого понятия, как внесение изме-

нений в коллекцию. Однако на уровне документа при помощи метода **update()**, основанного на полной замене совпавших документов, с оператором **$set** можно добавлять поля в документ, а с оператором **$unset** можно удалять поля из него. Общий вид оператора **update()**:

```
db.collection.update( <query>, <update>, <upsert>, <multi> )
```

Метод **update()** имеет следующие четыре параметра (табл. 7.1).

Таблица 7.1. **Параметры метода update()**

Параметр	Тип данных	Описание
query	document	Критерий выбора для обновления **Селекторы запросов** (query selectors) (см. приложение D)
update	document	Определяет изменения, которые необходимо внести. Если **<update>** документ содержит выражения оператора обновления, такие как использование оператора **$set**, тогда **<update>** документ должен содержать только выражения оператора обновления. Метод **update()** обновляет только соответствующие поля в документе. Если **<update>** документ содержит только поле: значение выражения, то: метод **update()** заменяет совпавший документ на **<update>** документа. Метод **update()** не заменяет значение **_id**
upsert	boolean	Не обязательный. Если установлено значение **true** и никакой из документов не соответствует заданному критерию, создается новый документ. Параметр **upsert** создает новый документ с полями и значениями либо **<update>** параметра, либо **<query>** и **<update>** параметров. Значение по умолчанию — **false**, которое не вставляет новый документ, если совпадений не обнаружено

Обратите внимание, что в приведенных ниже примерах используется значение параметра **multi: true**, поскольку изменения будут вноситься во все документы, соответствующие заданному критерию.

В соответствии с описанными выше параметрами метода произведем добавление поля **address** (для добавления адреса) в документы коллекции **personnel** (рис. 7.9):

```
db.personnel.update({}, { $set: { address: "" }},{ multi: true
})
```

Проверьте, что поле было вставлено (рис. 7.10).

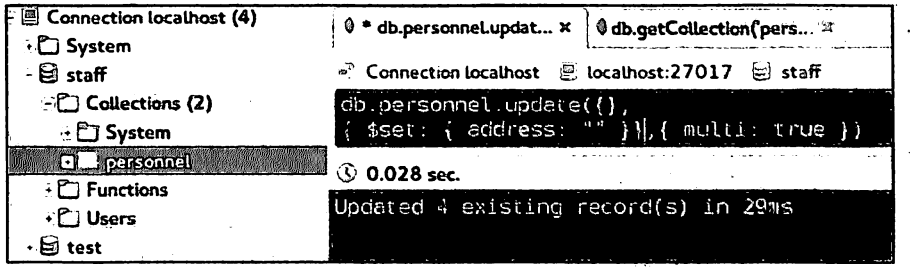


Рис. 7.9. Добавление поля **address** в документы коллекции

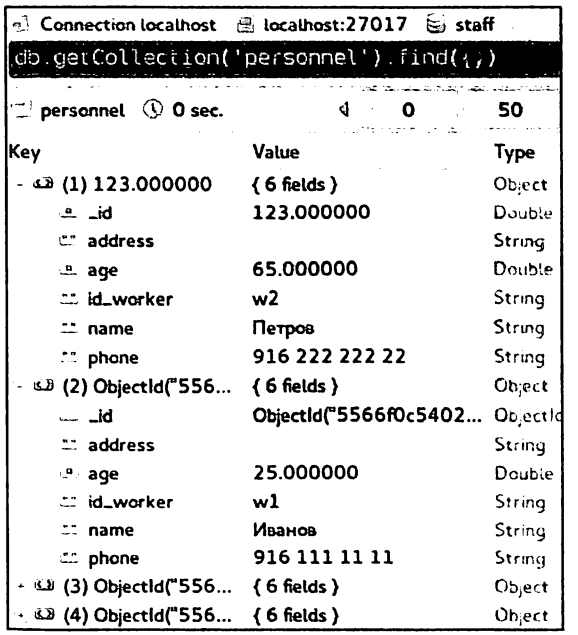


Рис. 7.10. Поле **address** добавлено в документы коллекции

Для того чтобы убрать вставленное поле, используется команда **$unset** (рис. 7.11):

```
db.personnel.update({}, { $unset: { address: "" }},{ multi:
true })
```

Для того чтобы проверить результат выполнения, снова отобразим коллекцию в виде дерева (рис. 7.12) или воспользуемся командой **db.personnel.find()**.

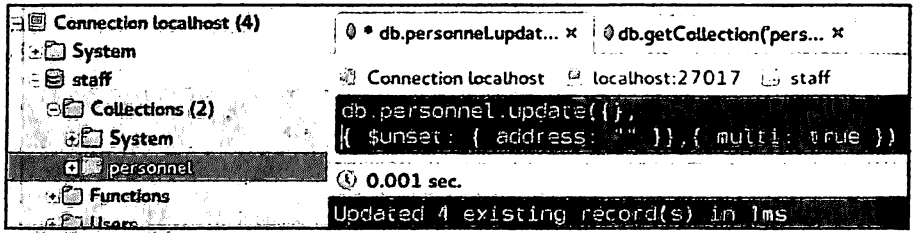


Рис. 7.11. Удаление поля **address** из документов коллекции

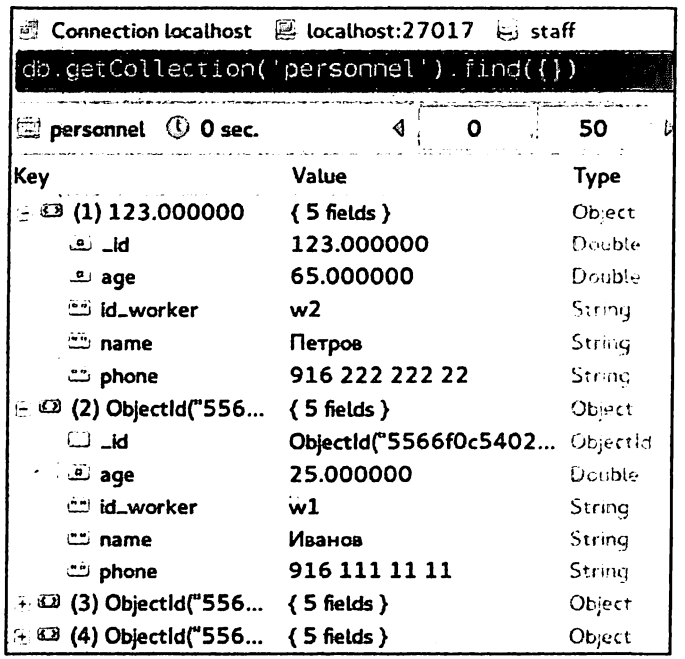


Рис. 7.12. Поле **address** удалено из документов коллекции

Как было сказано ранее, на языке SQL этим командам соответствуют команды

```
ALTER TABLE `staff`.`personnel` ADD COLUMN `address` TINYTEXT
NULL;
```

и

```
ALTER TABLE `staff`.`personnel` DROP COLUMN `address` ;
```

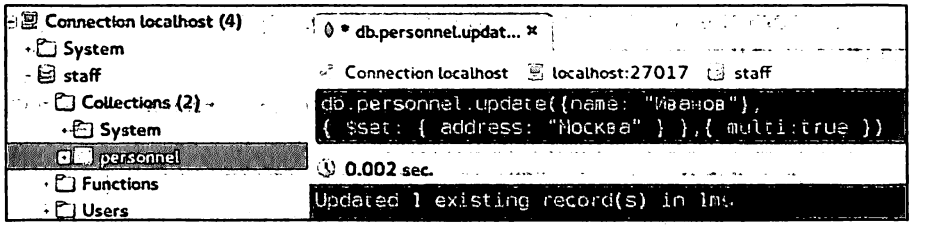



Рис. 7.13. Изменение в поле **address** в одном из документов коллекции

Предположим, необходимо внести **изменения в отдельные документы**, например изменить запись в адресе у Иванова (рис 7.13):

```
db.personnel.update({name: "Иванов"},{ $set: { address:
"Москва" } },{ multi: true })
```

Результат выполнения представлен на рис. 7.14.

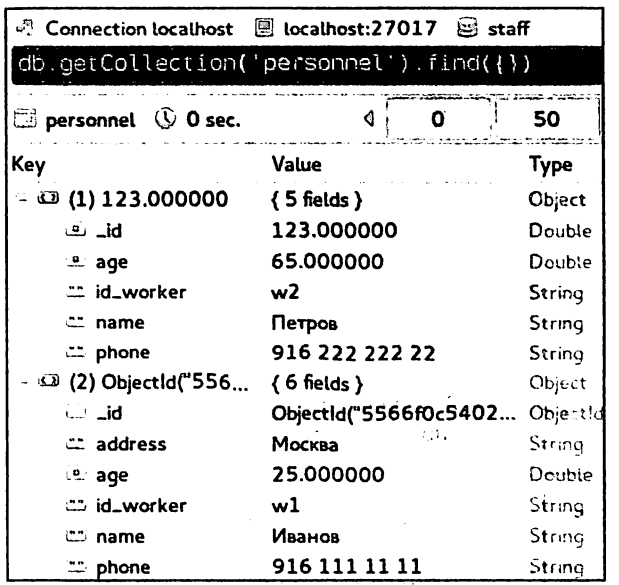


Рис. 7.14. Результат изменения в поле **address** в одном из документов коллекции

На языке SQL это соответствует оператору:

```
UPDATE staff.personnel SET address="Москва" WHERE
name="Иванов";
```

Аналогичным образом можно изменить номер телефона у Петрова (рис. 7.15):

```
db.personnel.update({name: "Петров"},{ $set: { phone: "916 987
76 54" } },{ multi: true })
```

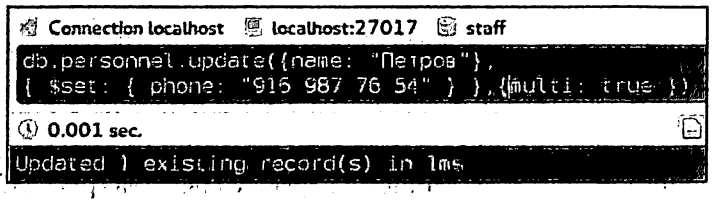


Рис. 7.15. Изменение номера телефона в одном из документов коллекции

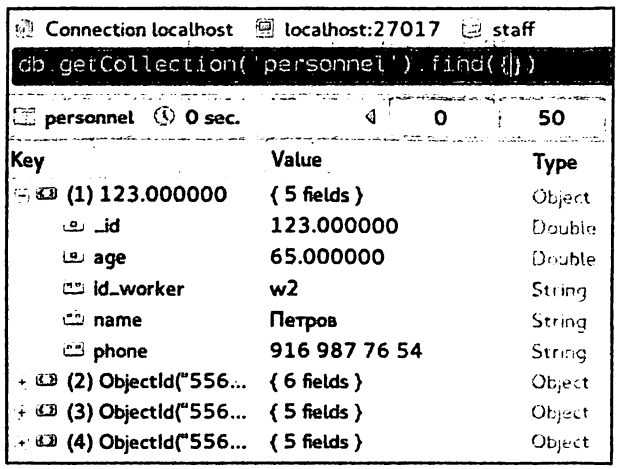


Рис. 7.16. Результат изменения номера телефона в одном из документов коллекции

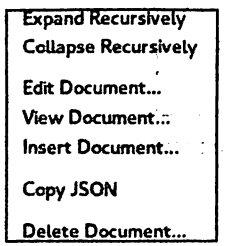


Рис. 7.17. Контекстное меню документов коллекции

Результат выполнения представлен на рис. 7.16.

Изменить значение поля при работе в оболочке Robomongo можно и другим способом. Например, изменим возраст сотрудника по фамилии Сидоров. Для этого необходимо вызвать контекстное меню документа в коллекции, щелкнув правой кнопкой мыши на выделенном документе (рис. 7.17). Контекстное меню позволяет сворачивать и разворачивать документ, его редактировать, осуществлять вставки и удалять документ.

После вызова пункта редактирования в контекстном меню для документа коллекции, содержащего поле **"name": "Сидоров"**, открывается окно редактирования. Заменим возраст Сидорова с 45 лет на 25 лет (рис. 7.18) и сохраним, нажав внизу экрана кнопку **«Save»**.

Проверим результат редактирования (рис. 7.19).

Если есть необходимость отредактировать отдельное поле документа, то также можно воспользоваться контекстным меню, которое

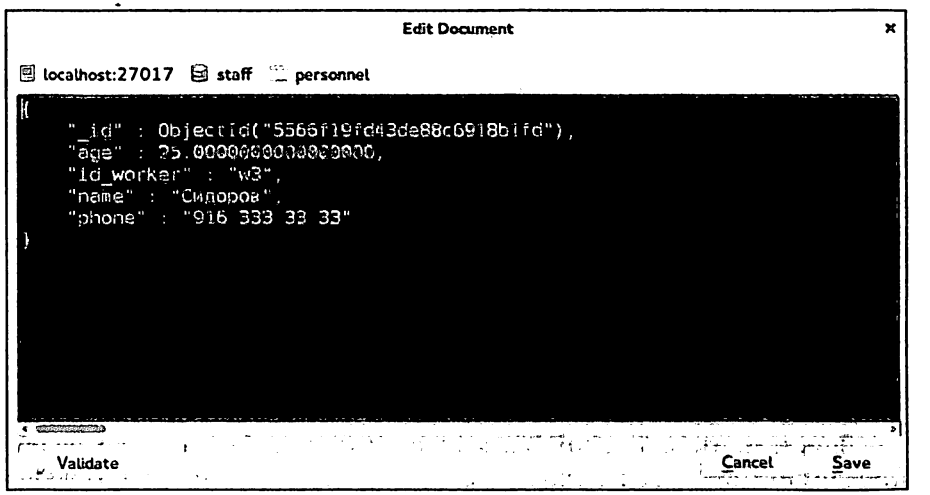


Рис. 7.18. Окно редактирования

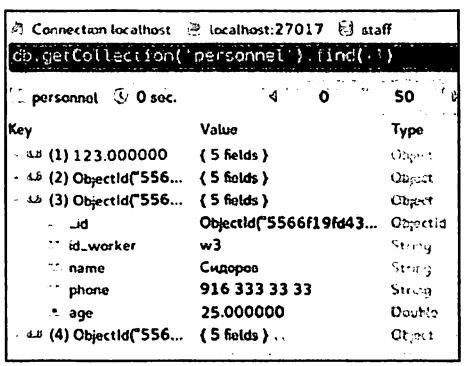


Рис. 7.19. Результат редактирования

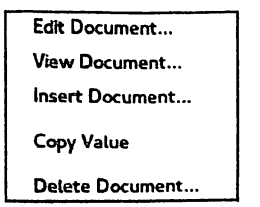


Рис. 7.20. Вариант контекстного меню

в случае вызова для конкретного поля имеет вид, как показано на рис. 7.20. В этом случае для вызова также необходимо щелкнуть правой кнопкой мыши на выделенном поле.

Заменим при помощи вызова данного варианта контекстного меню возраст Иванова на 55 лет (рис. 7.21).

Key	Value	Type
(1) 123.000000	{ 5 fields }	Object
(2) ObjectId("556...	{ 6 fields }	Object
_id	ObjectId("5566f0c5402...	ObjectId
address	Москва	String
age	55.000000	Double
id_worker	w1	String
name	Иванов	String
phone	916 111 11 11	String
(3) ObjectId("556...	{ 5 fields }	Object
_id	ObjectId("5566f19fd43...	ObjectId
age	25.000000	Double
id_worker	w3	String
name	Сидоров	String
phone	916 333 33 33	String
(4) ObjectId("556...	{ 5 fields }	Object

Рис. 7.21. Результат редактирования возраста Иванова

Заметим, что в MongoDB в запросах (в том числе запросах на обновление) могут быть использованы операторы сравнения: больше чем, меньше чем, больше или равно чем, меньше или равно чем (подробнее см. приложение D). Используя оператор меньше чем, заменим у людей, возраст которых 35 лет и менее, адрес на Ростов. Было произведено две замены, поскольку только у двоих число лет меньше или

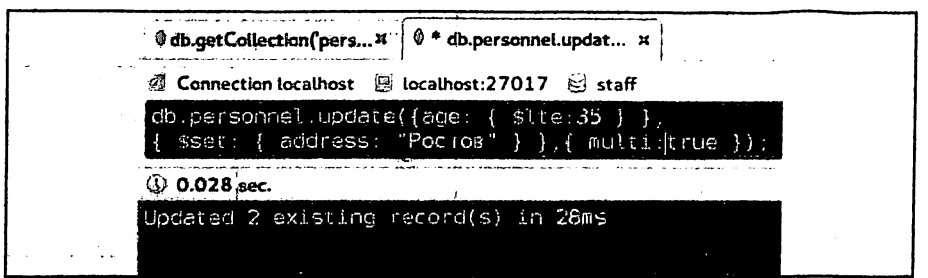
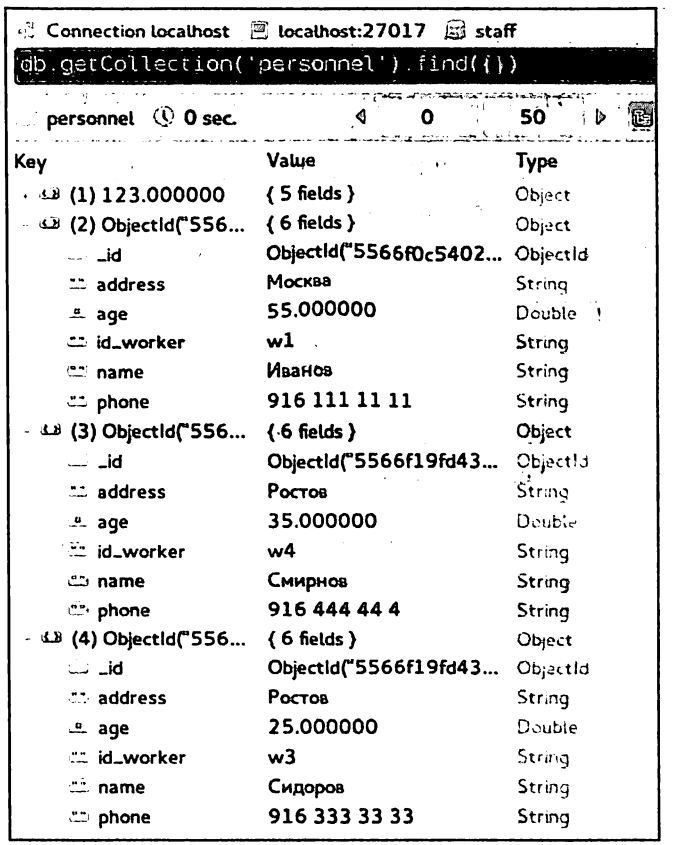


Рис. 7.22. Замена адреса, операторы **update()** и **$lte**

равно 35, а именно Смирнов имеет возраст 35 лет, Сидорову 25 лет (рис. 7.22):

```
db.personnel.update({age: { $lte:35 } },{ $set: { address:
"Ростов" } },{ multi: true });
```

Результат выполнения представлен на рис. 7.23.



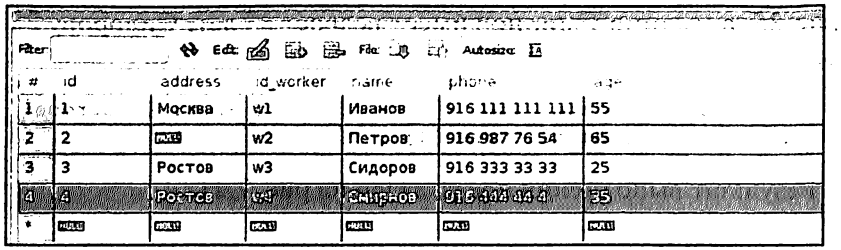
Key	Value	Type
(1) 123.000000	{ 5 fields }	Object
(2) ObjectId("556...	{ 6 fields }	Object
_id	ObjectId("5566f0c5402...	ObjectId
address	Москва	String
age	55.000000	Double
id_worker	w1	String
name	Иванов	String
phone	916 111 11 11	String
(3) ObjectId("556...	{ 6 fields }	Object
_id	ObjectId("5566f19fd43...	ObjectId
address	Ростов	String
age	35.000000	Double
id_worker	w4	String
name	Смирнов	String
phone	916 444 44 4	String
(4) ObjectId("556...	{ 6 fields }	Object
_id	ObjectId("5566f19fd43...	ObjectId
address	Ростов	String
age	25.000000	Double
id_worker	w3	String
name	Сидоров	String
phone	916 333 33 33	String

Рис. 7.23. Результат выполнения замены адреса, операторы **update()** и **$lte**

На языке SQL это соответствует оператору:

```
UPDATE staff.personnel SET address="Ростов" WHERE age<=35;
```

Результат для реляционной базы данных представлен на рис. 7.24.



#	id	address	id_worker	name	phone	age
1	1	Москва	w1	Иванов	916 111 111 111	55
2	2	NULL	w2	Петров	916 987 76 54	65
3	3	Ростов	w3	Сидоров	916 333 33 33	25
4	4	Ростов	w4	Смирнов	916 444 44 4	35
*	NULL	NULL	NULL	NULL	NULL	NULL

Рис. 7.24. Внесенные изменения в реляционной базе данных

Покажем, что количество полей документов может быть произвольным. Вставьте, например, данные о сотруднике по фамилии Соколов. Предположим, что иных данных, кроме его фамилии, нет (рис. 7.25).

```
db.personnel.insert({ name: "Соколов"});
```

Результат выполнения представлен на рис. 7.26.

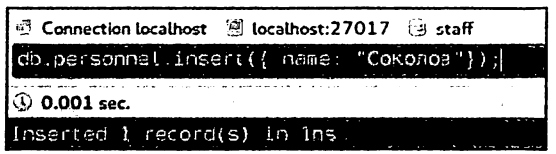


Рис. 7.25. Внесение данных, состоящих из одного поля

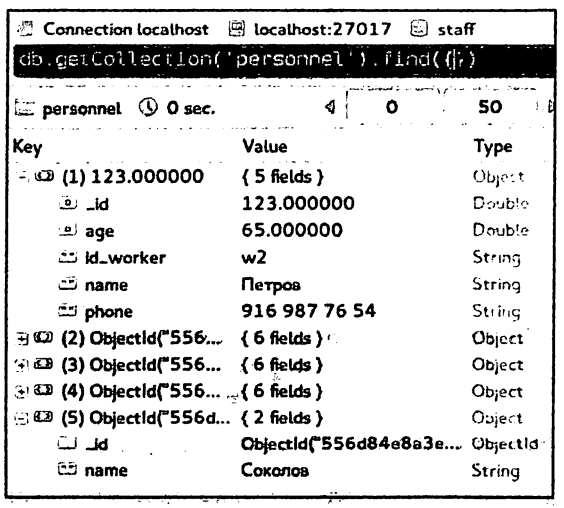


Рис. 7.26. Результат внесения данных, состоящих из одного поля

Внесем изменения в документ: пусть число лет Соколова равно 28 (рис. 7.27).

```
db.personnel.update({name: "Соколов"},{ $set: { age: 28} });
```

Результат выполнения представлен на рис. 7.28.

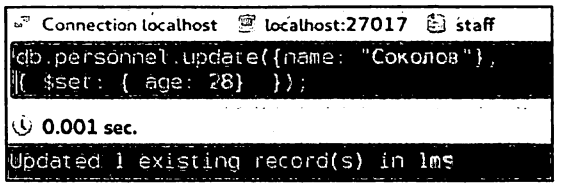


Рис. 7.27. Внесение данных поля возраст

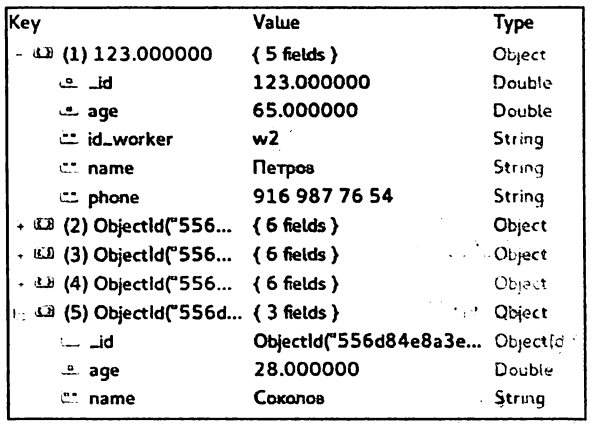


Рис. 7.28. Результат внесения данных поля возраст

Далее рассмотрим внесение изменений с параметром **upsert**. Заметим, что при использовании **upsert**, если документ по требуемому критерию не найден, то он будет создан, если же найден, то он будет обновлен, как обычно. Например, изменим документ, создадим дополнительное поле — добавим хобби Соколову. Сделаем это с двумя разными значениями параметра **upsert**. В первом случае значение параметра будет **false** (рис. 7.29).

```
db.personnel.update({name: "Соколов"},{ $set: { hobby:
"fishing"} },{ upsert: false });
```

Результат выполнения представлен на рис. 7.30.

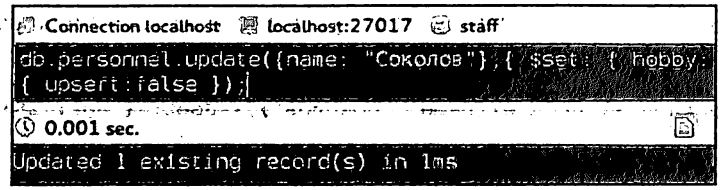


Рис. 7.29. Внесение данных поля хобби

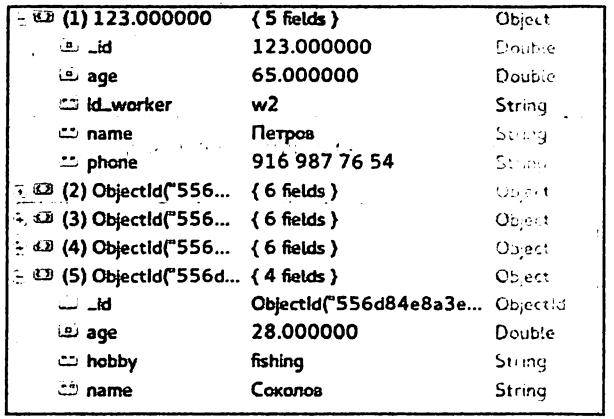


Рис. 7.30. Результат внесения данных поля хобби

Добавим второе хобби Соколову. Сделаем со значением параметра **true** (рис. 7.31):

```
db.personnel.update({name: "Соколов"},{ $set: { hobby2:
"alpinism"} },{ upsert: true });
```

Результат выполнения (рис. 7.32).

Таким образом, вне зависимости от значения параметра **upsert** (**true** или **false**), при нахождении совпадения документ будет обновлен, как обычно.

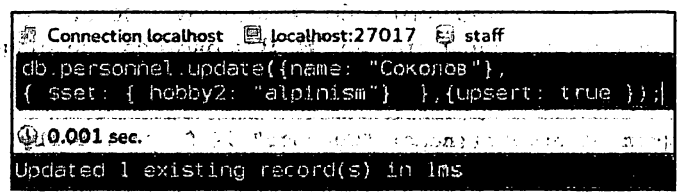


Рис. 7.31. Внесение данных поля второе хобби

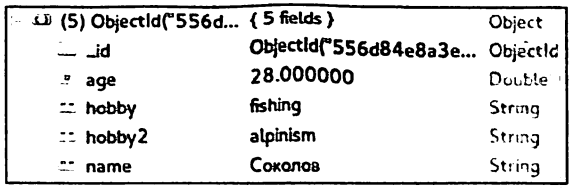


Рис. 7.32. Результат внесения данных поля второе хобби

Теперь рассмотрим случай, когда совпадений заведомо не будет, а значение параметра **upsert: false**. Попробуем внести данные о Федорове (рис. 7.33):

```
db.personnel.update({name: "Федоров"},{ $set: { hobby2:
"alpinism"} },{ upsert: false });
```

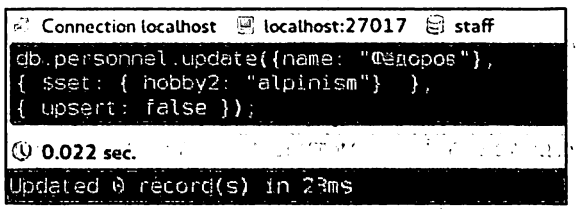


Рис. 7.33. Внесение данных поля хобби **upsert: false**

Поскольку совпадений не обнаружено, запись не была внесена, что следует из сообщения в нижней части окна: **Updated 0 record(s)**.

Изменим значение параметра **upsert: true** (рис. 7.34):

```
db.personnel.update({name: "Федоров"},{ $set: { hobby2:
"alpinism"} },{ upsert: true });
```

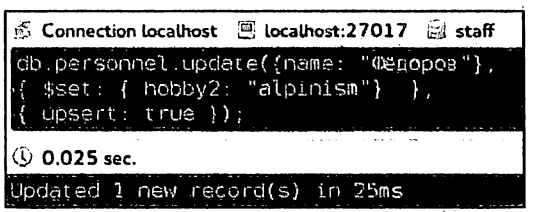


Рис. 7.34. Внесение данных поля хобби **upsert: true**

Результат выполнения представлен на рис. 7.35.

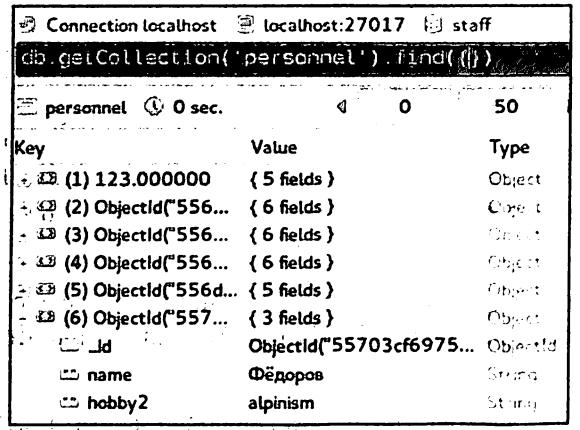


Рис. 7.35. Результат внесения данных поля хобби **upsert: true**

Напоминаем, что во всех приведенных выше примерах было использовано значение параметра **multi: true**, поскольку требовалось внести изменения во все документы, соответствующие заданному критерию. Если данный параметр принимает значение **false**, то ищется первый документ, соответствующий критерию, и он обновляется. Рассмотрим пример обновления поля **address** у людей старше 40 лет (рис. 7.36):

```
db.personnel.update({age: { $gt:40 } },{ $set: { address:
"Саратов" } },{ multi: false });
```

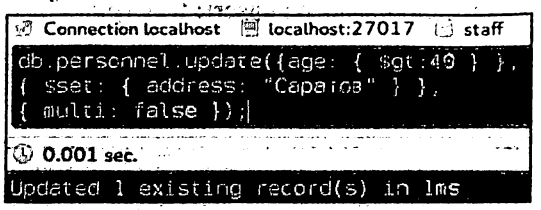


Рис. 7.36. Обновление поля **address** с параметром **multi: false**

Обратите внимание, что была обновлена только одна (первая найденная запись). Результат выполнения представлен на рис. 7.37.

Удаление документов из коллекции происходит при помощи метода **remove()**. С его помощью можно удалить все документы из коллекции либо один или несколько документов, соответствующих заданному условию. Для удаления всех документов коллекции следует выполнить команду

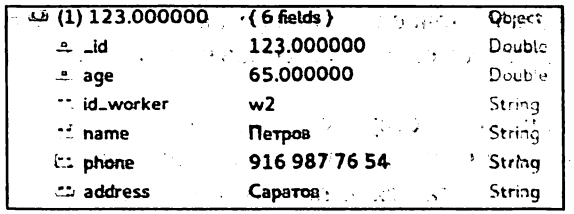


Рис. 7.37. Результат изменения поля **address** с параметром **multi: false**

```
db.personnel.remove();
```

Необходимо заметить, что ни сама коллекция, ни индексы из нее не будут удалены.

Если необходимо удалить документ, то нужно задать условие выборки, например имя. Читатель может самостоятельно добавить документ с данными сотрудника по фамилии Климов при помощи команды

```
db.personnel.insert({name:"Климов"});
```

А затем этот документ удалить при помощи команды

```
db.personnel.remove({name: " Климов " });
```

или даже

```
db.personnel.remove({name: /^Кли/ });
```

Принципы выборки полей из документа и их сортировка описаны ниже.

Для того чтобы полностью удалить коллекцию, используйте метод **drop()**. Создайте коллекцию **p1**, внесите туда некоторый документ:

```
db.p1.insert({name:"Климов"});
```

А затем удалите:

```
db.p1.drop();
```

Для удаления базы данных следует сначала убедиться, что именно эта база данных является используемой, а затем удалить ее:

```
use mydb;
db.dropDatabase();
```

При работе в оболочке Robomongo для удаления документов, коллекций и баз данных можно воспользоваться контекстным меню (см. выше).

Очевидно, что существует еще значительное количество команд вставки и обновления данных (например, **save**, **push**, **pop**, **pull** и др.), их использование предлагается читателю освоить самостоятельно.

Учащимся предлагается выполнить практическую работу, основанную на изученном теоретическом материале. Выполнение практической работы предполагает освоение следующих операций: создание, обновление и удаление документов в коллекции СУБД MongoDB.

Порядок выполнения практической работы

1. Ознакомиться с описанием.
2. Запустить оболочку **Robomongo**.
3. Получить у преподавателя задание на создание базы данных для некоторой предметной области.
4. Определить основные поля документа разрабатываемой базы данных.
5. Создать документы, внося данные, определив в некоторых документах поле **_id** самостоятельно.
6. Модифицировать данные, добавив или удалив столбцы в соответствии с заданием.
7. Заменить при помощи контекстного меню значение поля в соответствии с заданием.
8. Заменить значение полей при помощи оператора **update( )** с параметром **multi: true**.
9. Удалить один из документов коллекции.
10. Сохранить базу данных и выйти из программы.
11. Подготовить отчет по практической работе, содержащий основные этапы работы с базой данных в оболочке **Robomongo**.

Контрольные вопросы

1. Аналогом какого оператора языка SQL является функция **find()**?
2. Для каких целей служит метод **insert()** в **MongoDB**?
3. Для чего служит поле **_id** в **MongoDB**?
4. Каким образом можно создать в поле **_id** уникальный **ObjectId** документа?

5. При помощи какого метода можно удалить все документы из коллекции, а также один или несколько документов, соответствующих заданному условию?
6. Какой метод используется для изменения коллекций? Какие параметры он имеет?
7. Какому методу в **MongoDB** соответствует команда **ALTER TABLE** в языке SQL?
8. Каким образом вносятся изменения в отдельные документы?
9. В чем отличие работы метода **update()** при значении параметра **multi: true** от его работы при значении параметра **multi: false**?
10. Какие параметры метода **db.collection.find()** вы знаете? Для чего они используются?

Лабораторная работа 8
ВЫБОРКА ДАННЫХ ИЗ КОЛЛЕКЦИЙ

Цель: освоение методов выборки данных из коллекций MongoDB.

Для осуществления выборки из коллекции (аналог операции **SELECT** в реляционной базе данных) используется метод **db.collection.find(<criteria>, <projection>)**, представленный в табл. 8.1.

Таблица 8.1. **Параметры метода db.collection.find()**

Параметр	Тип данных	Описание
criteria	Документ	Необязательный. Определяет критерий выбора с использованием операторов запроса. Для возврата всех документов коллекции следует опустить данный параметр или выбрать пустой документ ({})
projection	Документ	Необязательный. Определяет поля, которые будут возвращены. Для возврата всех полей заданного документа данный параметр следует опустить

Параметр **projection** принимает документ следующего вида:

```
{ field1: <boolean>, field2: <boolean> ... }
```

Значение **<boolean>** может быть следующим:

- **1** или **true** для включения поля (метод **find()** по умолчанию всегда возвращает поле **_id**, даже если оно явно не указано; чтобы исключить его, необходимо явным образом указать **{field:1, _id: 0}**);
- **0** или **false** для исключения поля.

Рассмотрим выбор двух полей: возраст и адрес (в первом случае поле **_id** явно не исключено, поскольку всегда включается в выборку автоматически):

```
db.personnel.find({}, {age:1, address:1});
```

Результат выполнения представлен на рис. 8.1 (поле **_id** присутствует).

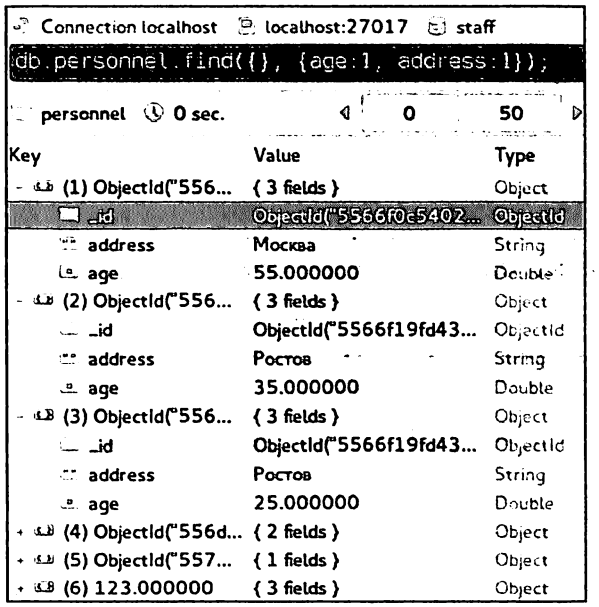


Рис. 8.1. Отображение полей «возраст» и «адрес», поле **_id** присутствует

Во втором случае поле **_id** явно исключено (**_id:0**):

```
db.personnel.find({}, {age:1, address:1, _id:0});
```

Результат выполнения представлен на рис. 8.2 (поле **_id** отсутствует).

Аналогичный результат в реляционной базе данных (например, MariaDB) можно достичь, если выполнить оператор **SELECT** с перечислением выводимых полей. Примеры выполнения SQL-запросов, один из которых выводит поле **id**, а второй не выводит данное поле, приведены на рис. 8.3 и 8.4 соответственно.

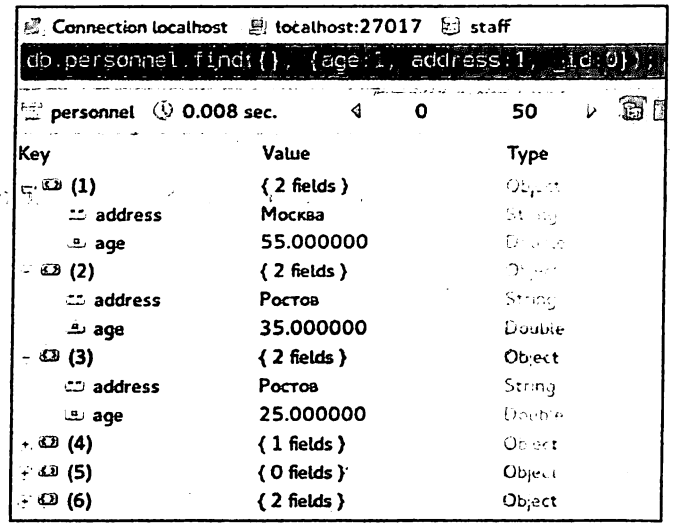


Рис. 8.2. Отображение полей «возраст» и «адрес», поле _id отсутствует

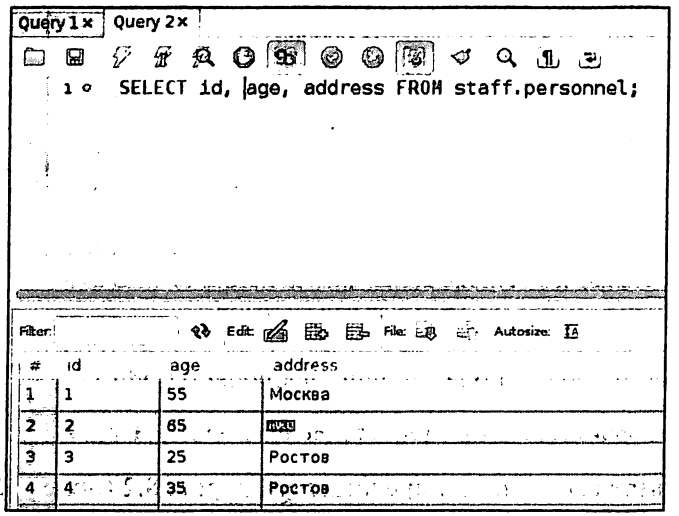


Рис. 8.3. Результат выполнения SQL-запроса, выводящего поле id

Рассмотрим выполнение запросов на выборку в MongoDB (через Robomongo), содержащих не только задание полей в явном виде, но и условные операторы.

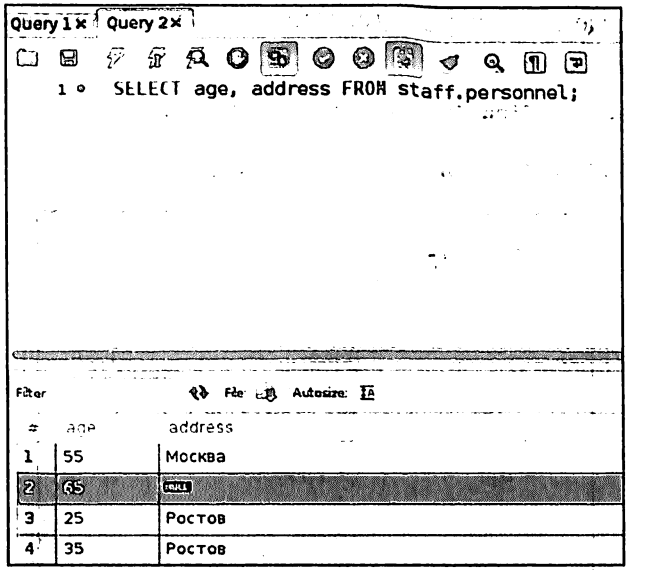


Рис. 8.4. Результат выполнения SQL-запроса, который не выводит поле id

Произведем выборку документов, в которых фамилия сотрудника Петров, при помощи команды:

```
db.personnel.find({name:"Петров"});
```

Результат выполнения представлен на рис. 8.5.

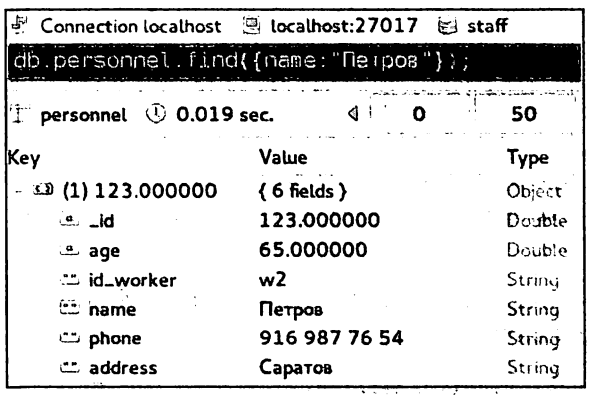


Рис. 8.5. Результат выборки документов с именем «Петров»

Произведем выборку документов, в которых фамилия сотрудника не Петров.

```
db.personnel.find({name:{$ne:"Петров"}});
```

Результат выполнения представлен на рис. 8.6.

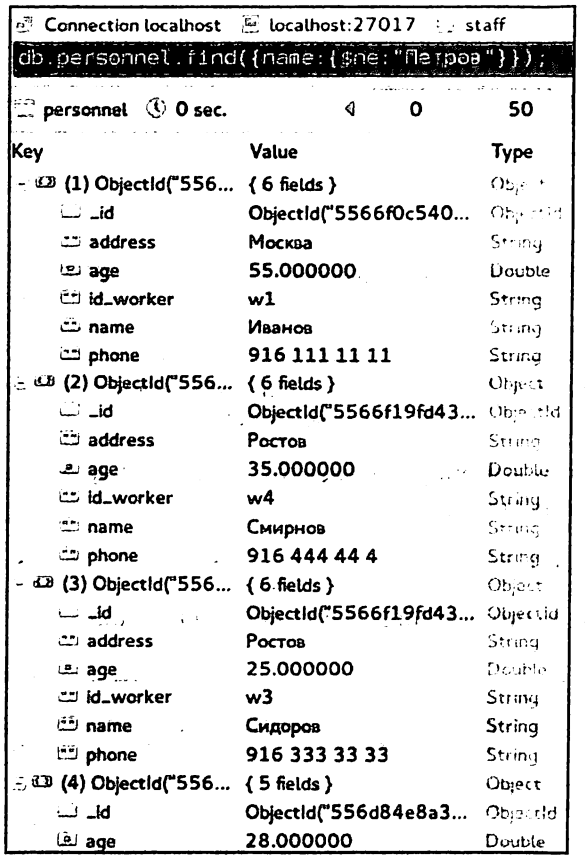


Рис. 8.6. Результат выборки документов с именем «не Петров»

Для поиска подстрок используется аналог SQL-оператора **LIKE**, который имеет вид:

```
db.personnel.find({name:/етр/});
```

Результат выполнения поиска по части имени (рис. 8.7).

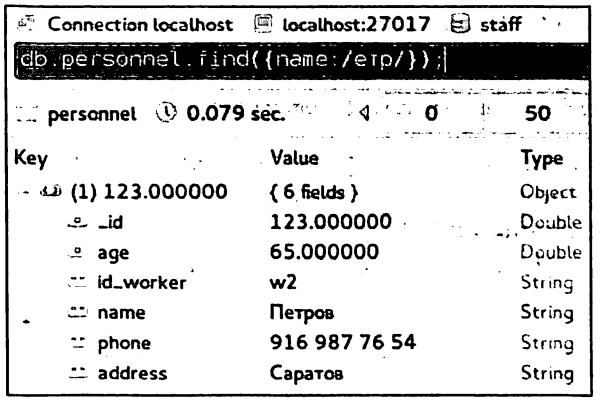


Рис. 8.7. Результат выборки документа по части имени

Или поиск по началу фамилии:

```
db.personnel.find({name:/^Петр/});
```

Результат выполнения поиска по началу имени (рис. 8.8).

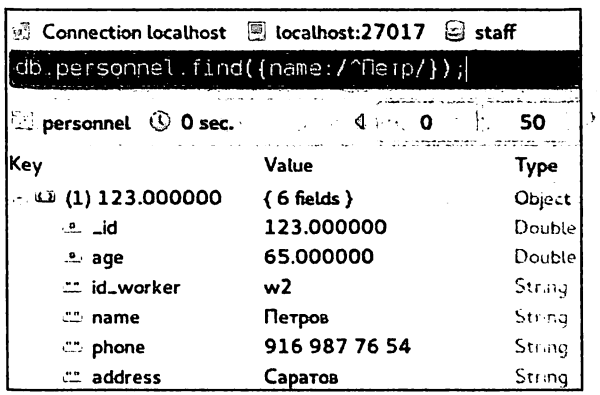


Рис. 8.8. Результат выборки документа по началу имени

Обратите внимание, что во втором случае в запросе использовалось регулярное выражение. Тема регулярных выражений ввиду ее обширности выходит за рамки данной книги. Заметим только, что также существует оператор **$regex**, который задает регулярное выражение. Этому выражению должно соответствовать значение поля.

Например, для поиска имени, содержащего букву «р», можно использовать следующий код:

```
db.personnel.find({name: {$regex:"р"});
```

Аналогичный результат в реляционной базе данных можно достичь, если выполнить оператор **SELECT** с ключевым словом **LIKE**. Примеры выполнения SQL-запросов, один из которых производит поиск по части фамилии, а второй — по началу, приведены на рис. 8.9 и 8.10 соответственно.

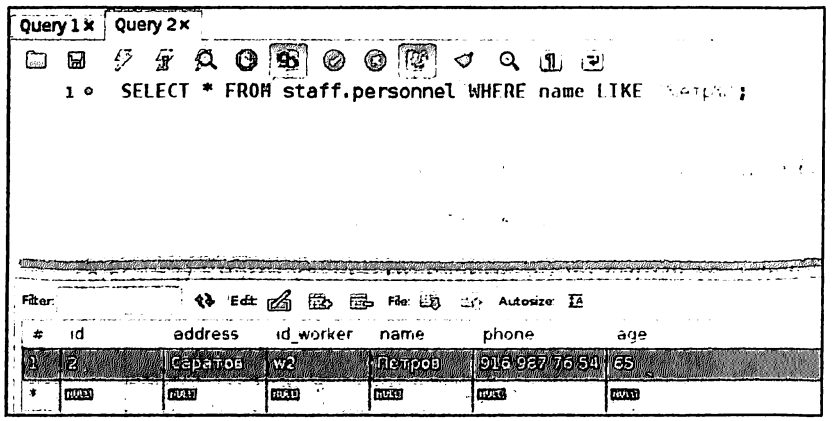


Рис. 8.9. Результат выполнения SQL-запроса, производящего поиск по части фамилии

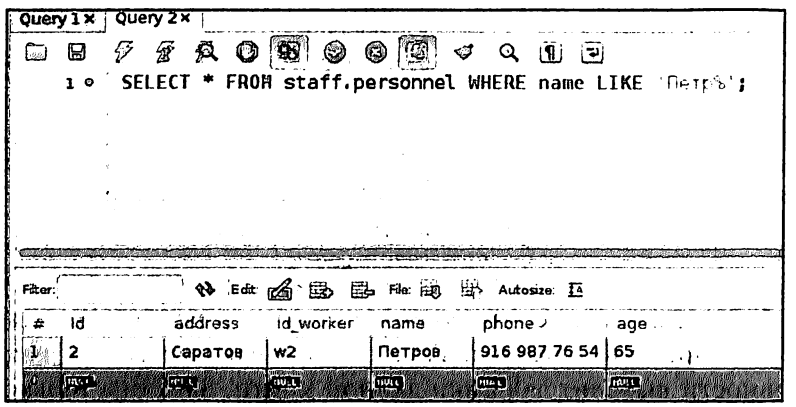


Рис. 8.10. Результат выполнения SQL-запроса, производящего поиск по началу фамилии

Для сортировки данных в MongoDB используется метод **sort()**. Если значение для поля рано **1**, то сортировка производится по возрастанию, а если **-1**, то по убыванию. Рассмотрим выполнение примеров сортировки фамилий. В первом случае сортировка выполняется в алфавитном порядке от А до Я, во втором — наоборот (рис. 8.11):

```
db.personnel.find({},{name:1, _id:0}).sort({name:1});
```

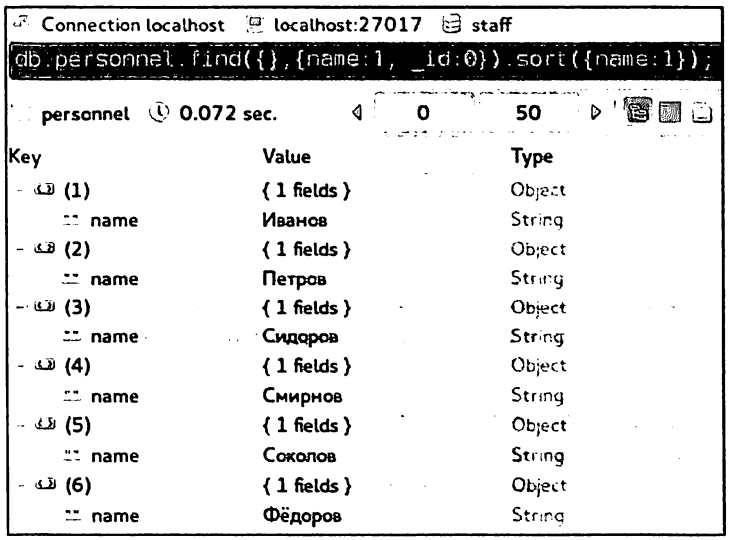


Рис. 8.11. Результат сортировки по алфавиту от А до Я

Для сортировки в обратном порядке от Я до А поле равно **-1** (рис. 8.12).

```
db.personnel.find({},{name:1, _id:0}).sort({name:-1});
```

Аналогичный результат в реляционной базе данных можно достичь, если выполнить оператор **SELECT** с ключевым словом **ORDER BY**. Примеры выполнения SQL-запросов, осуществляющих прямую и обратную сортировки, приведены на рис. 8.13 и 8.14 соответственно.

Для подсчета числа документов, удовлетворяющих некоторому условию, в MongoDB используется метод **count()**. Например, чтобы определить число сотрудников, т. е. количество документов с записью в поле **name**:

```
db.personnel.count({name:{$exists:true}});
```

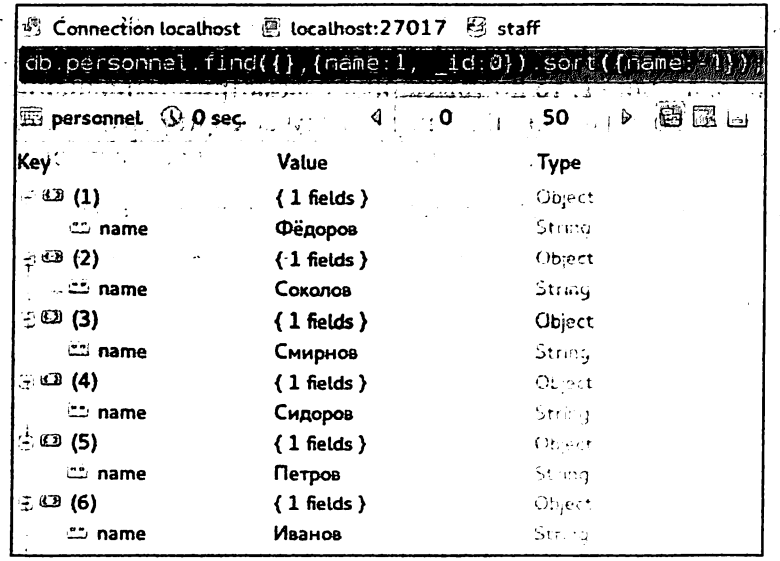


Рис. 8.12. Результат сортировки от Я до А

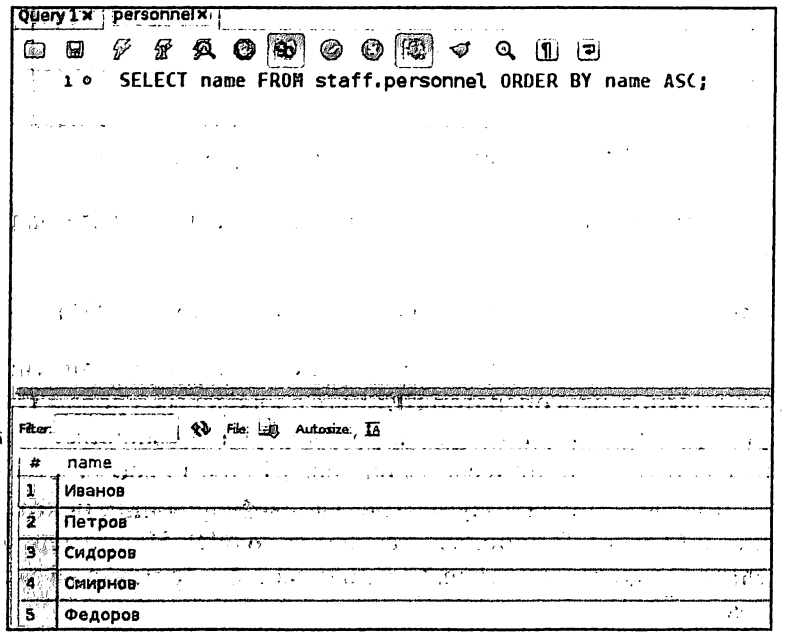


Рис. 8.13. Результат выполнения прямой сортировки

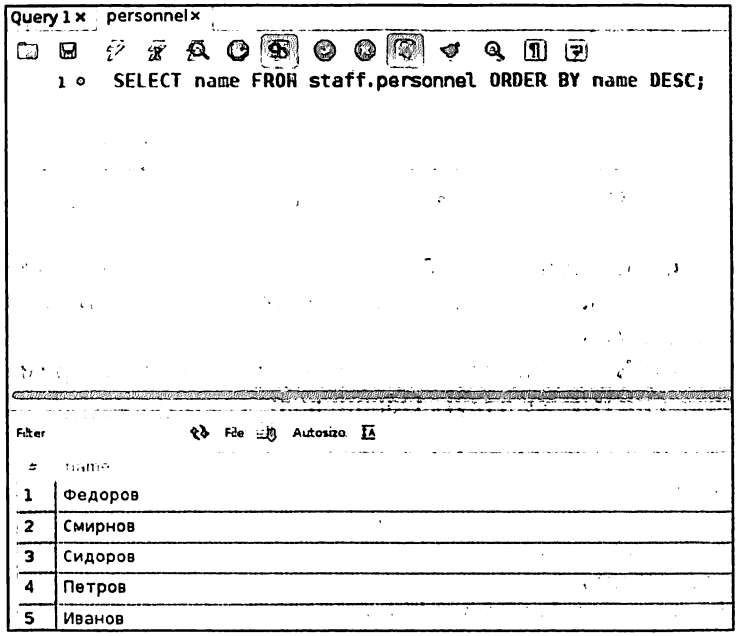


Рис. 8.14. Результат выполнения обратной сортировки

Результат выполнения (рис. 8.15) говорит о том, что имеется шесть документов, у которых есть записи в поле **name**.

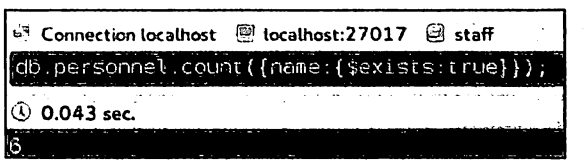


Рис. 8.15. Подсчет числа документов с помощью оператора **count()**

Чтобы определить количество документов с записью в поле **address**, следует выполнить команду:

```
db.personnel.count({address:{$exists:true}});
```

Результат выполнения говорит о том, что имеется четыре документа, у которых есть записи в поле **address** (рис. 8.16).

В некоторых случаях при работе с данными полезно использовать оператор группировки (в языке SQL **GROUP BY** применяется для

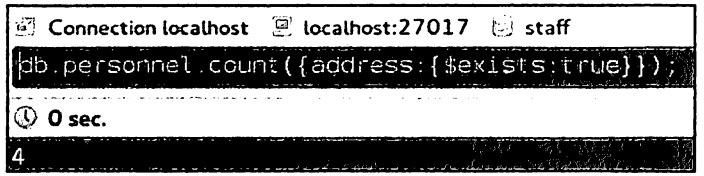


Рис. 8.16. Подсчет числа документов с записью в поле **address**

объединения результатов выборки по одному или нескольким столбцам, и с ним, как правило, связаны агрегатные функции: **COUNT**, **MIN**, **MAX**, **AVG** и пр.).

В **MongoDB** использование метода **group** аналогично применению выражения **GROUP BY** в SQL. Общий вид:

```
{
 group:
 {
   ns: <namespace>,
   key: <key>,
   $reduce: <reduce function>,
   $keyf: <key function>,
   cond: <query>,
   finalize: <finalize function>
 }
}
```

Метод **group** принимает следующие параметры (табл. 8.2).

Таблица 8.2. **Параметры метода group**

Поле	Тип данных	Описание
ns	string	Коллекция, для которой выполняется оператор группировки
key	document	Указывает на ключ, по которому надо проводить группировку
$reduce	function	Функция агрегации, обрабатывающая документы для группировки, также может возвращать сумму или подсчитывать число документов. Имеет два аргумента: текущий документ и документ с результатами агрегации. В текущей версии может быть использована без знака $

Окончание табл. 8.2

Поле	Тип данных	Описание
initial	document	Описывает и инициализирует документ для хранения результата агрегации
$keyf	function	Необязательный параметр. Является альтернативой параметру **key**. Определяет функцию, которая создает «**key object**» для использования в качестве ключа для группировки. **$keyf** следует использовать вместо параметра **key** для группировки вычисленных полей, а не для уже имеющихся в документе полей
cond	document	Необязательный параметр. Предназначен для выбора критерия для определения, какой из документов коллекции подлежит группировке. Если данный параметр пропущен, то в группировке участвуют все документы
finalize	function	Необязательный параметр. Представляет функцию, которая запускается каждый раз перед тем, как будут возвращены результаты группировки. Данная функция может как модифицировать результаты группировки, так и поменять результат полностью

Рассмотрим пример группировки по одному полю — полю **address** (рис. 8.17):

```
db.personnel.group(
{
  key: { address: 1 },
  reduce: function( curr, result ) { result.total += 1; },
  initial: { total : 0 }
}
);
```

В результате сгруппированы записи: одна с городом Москва, две с городом Ростов, две с пустым полем и одна с городом Саратов, что соответствует значениям полей **address** в документах коллекции.

Если необходимо провести группировку с некоторым условием, используется параметр **cond**. Сгруппируем по адресу людей старше 45 лет. Результат выполнения представлен на рис. 8.18. Очевидно, что

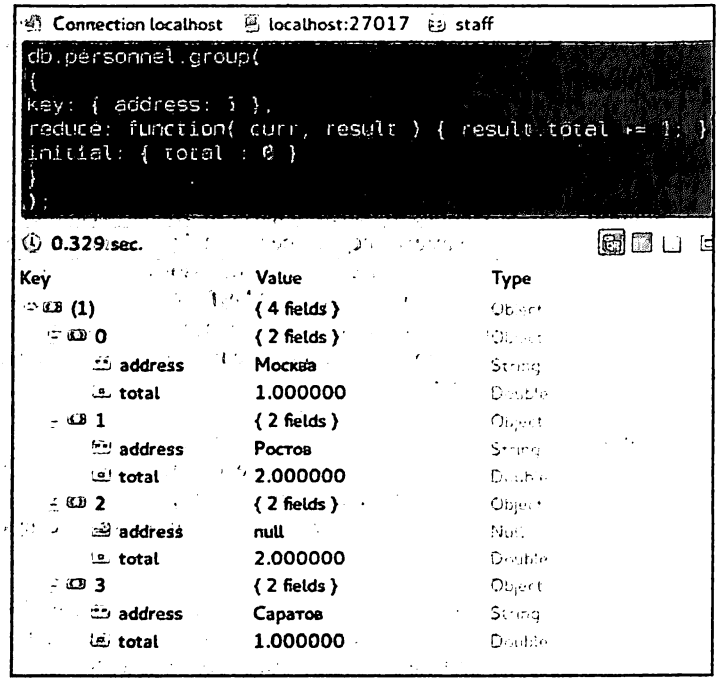


Рис. 8.17. Результат выполнения группировки по полю **address**

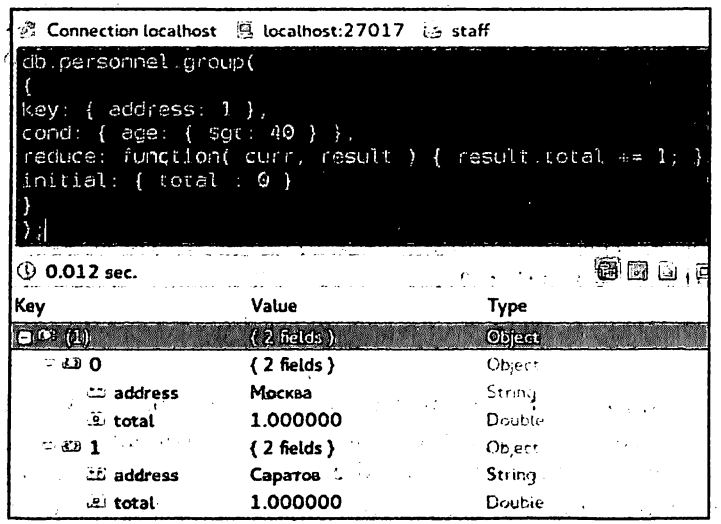


Рис. 8.18. Результат выполнения группировки с условием по полю **address**

два документа (соответствующие возрасту Иванова и Петрова) подлежат группировке при наличии соответствующего условия **cond: {age:{ gt: 40 }}**. Заметим, что если поле возраста пустое, то документ в группировке не участвует.

```
db.personnel.group(
{
  key: { address: 1 },
  cond: { age: { $gt: 40 } },
  reduce: function( curr, result ) { result.total += 1; },
  initial: { total : 0 }
}
);
```

Если поля не заполнены, например, у Федорова отсутствует поле возраст (**age**), то при наличии условия для этого поля данный документ в группировку не попадает (рис. 8.19). В данном примере в группировку по адресу «Ростов» не попадает Сидоров, поскольку его возраст не соответствует условию больше 25 лет, а документ с именем

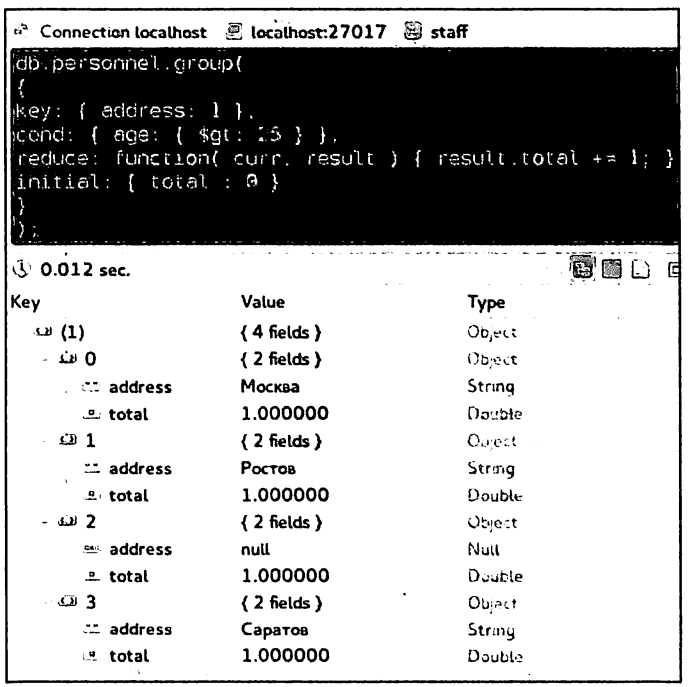


Рис. 8.19. Результат выполнения группировки с условием по полю **address** для пустых полей

Федоров не участвует в группировке, поскольку у него поле возраста отсутствует:

```
db.personnel.group(
{
  key: { address: 1 },
  cond: { age: { $gt: 25 } },
  reduce: function( curr, result ) { result.total += 1; },
  initial: { total : 0 }
}
);
```

Приведем еще один пример группировки, которому соответствует использование агрегатной функции **SUM** в языке SQL. В данном примере помимо группировки по адресу людей моложе 20 лет вычисляется их суммарный возраст. Результат представлен на рис. 8.20. В данном случае суммирование будет производиться по адресу «Ростов» и по полю, которое не заполнено, поскольку именно в эту группировку попадает по два человека. Для группировки «Ростов» имеем двух людей: Сидоров 25 лет и Смирнов 35 лет. По незаполненному полю адрес имеем также Соколова 28 лет и Федорова, у которого возраст не заполнен. Соответственно, в группировке по полю «Ростов» суммарный возраст будет равен 60, а по незаполненному полю — 28.

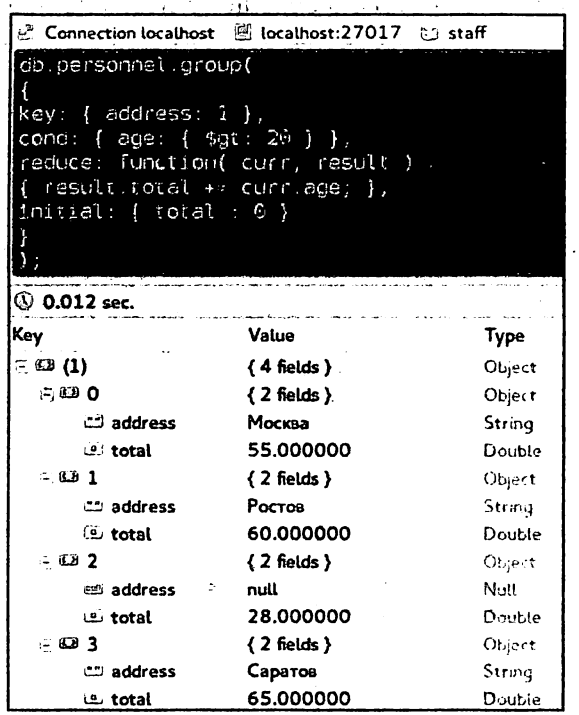


Рис. 8.20. Результат выполнения группировки с условием по полю **address** и суммированием

```
db.personnel.group(
{
  key: { address: 1 },
  cond: { age: { $gt: 20 } },
  reduce: function( curr, result )
  { result.total += curr.age; },
  initial: { total : 0 }
}
);
```

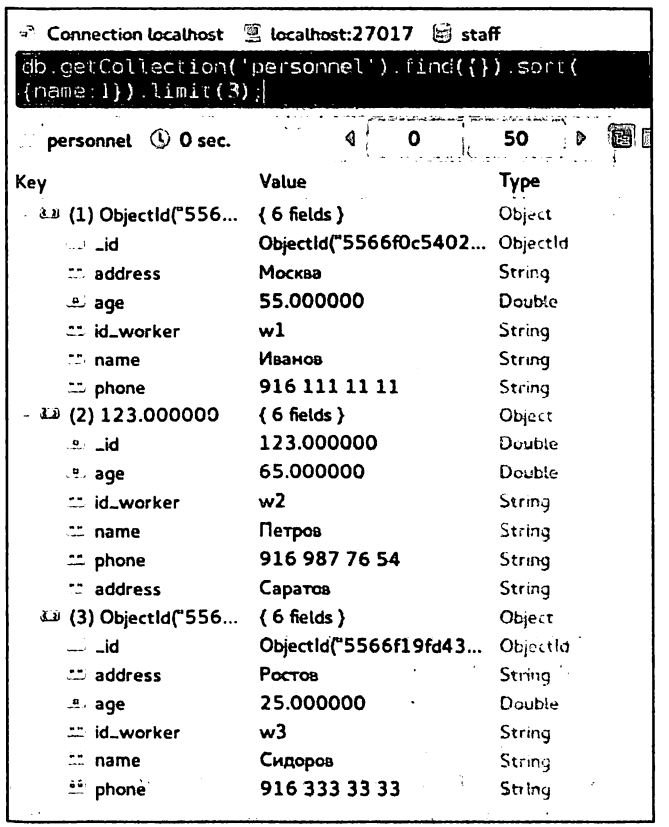


Рис. 8.21. Результат ограничения вывода числа записей

Для того чтобы получить ограниченное количество документов по запросу, используется команда **limit()** (рис. 8.21), которая в данном примере ограничит вывод документов первыми тремя (в данном примере коллекции предварительно отсортированы по полю имени по алфавиту):

```
db.getCollection('personnel').find({}).limit(3);
```

Для того чтобы пропустить несколько документов и получить все остальные, используется команда **skip()** (рис. 8.22), в результате выполнения которой будут отображены три документа, начиная с четвертого:

```
db.getCollection('personnel').find({}).skip(3).limit(3);
```

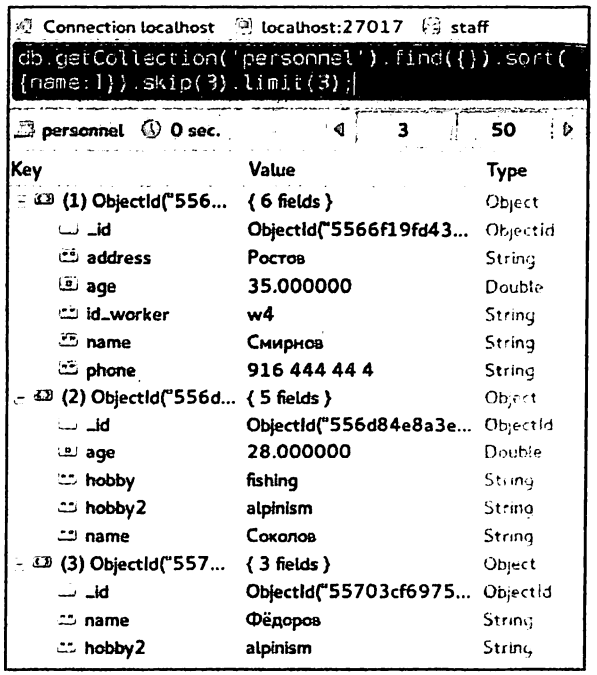


Рис. 8.22. Результат пропуска некоторого числа записей

Изучение остальных команд для работы с запросами, таких как **findOne()** (возврат первого совпавшего документа), **$slice** (для поиска документов из середины) и пр., предоставляется в качестве самостоятельного упражнения.

При внесении данных в базу тип данных (см. примеры выше) определяется автоматически. Если число было введено не как 123, а как «123», то он будет интерпретироваться как строка (что совпадает с тем, как это делают интерпретируемые языки Perl, PHP, JavaScript). Однако при работе с документами иногда необходимо иметь возможность явно задать тип данных конкретного поля. В следующем примере внесем данные о возрасте в документ с именем «Федоров» (рис. 8.23).

```
db.personnel.update({name:
"Федоров"},{set:{age:NumberLong("67")}});
```

Результат выполнения представлен на рис. 8.24.

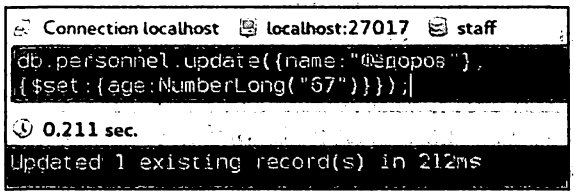


Рис. 8.23. Задание типа данных в явном виде

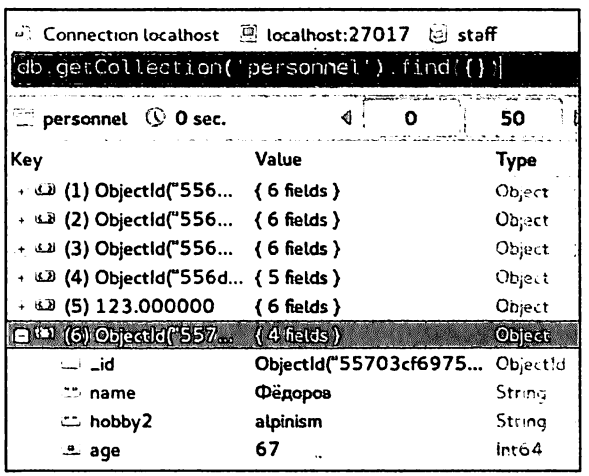


Рис. 8.24. Отображение явно заданного типа данных в коллекции

Однако в самом документе тип данных будет иметь вид, как показано на рис. 8.25.

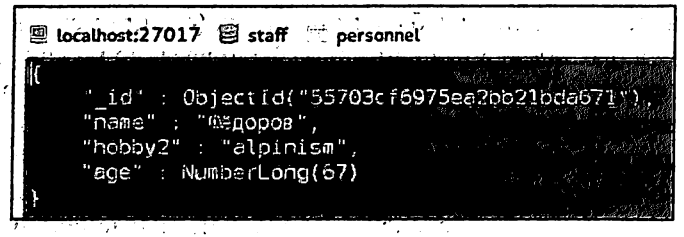


Рис. 8.25. Отображение явно заданного типа данных в документе

Поскольку тип данных конкретного поля в NoSQL СУБД не регламентируется в отличие от реляционных СУБД, то иногда при заполнении данных в конкретное поле в разных документах могли быть внесены данные различных типов. Поэтому бывает необходимо выбрать конкретное поле (поля) с конкретным типом данных. Для этого каждому типу данных в MongoDB ставится в соответствие некоторое число, например, числу типа double соответствует 1, строке (string) 2 и т. п. Подробнее см. приложение С. Выберем все документы, у которых в поле **name** строковые данные (очевидно, что это все документы (рис. 8.26)):

```
db.personnel.find( {name: { $type : 2 } } );
```

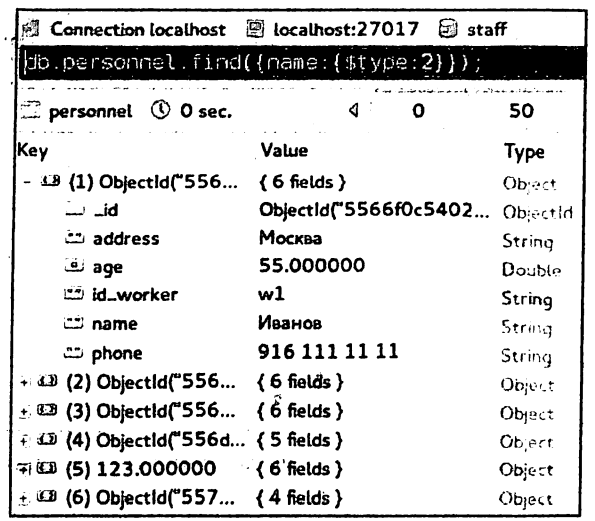


Рис. 8.26. Выбор документов, у которых в поле **name** тип данных строка

Для сравнения выберем все документы, у которых в поле **name** число (очевидно, что не найдется ни одного документа (рис. 8.27)):

```
db.personnel.find( {name: { $type : 1 } } );
```

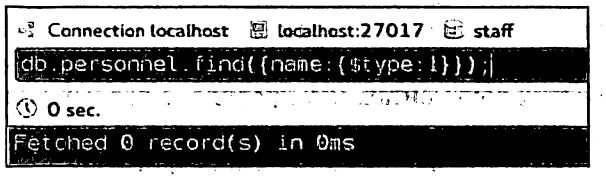


Рис. 8.27. Выбор документов, у которых в поле **name** тип данных число типа double

Однако если мы сделаем выборку по полю **age**, то получим только пять записей. Запись с именем «Федоров» туда не попадет, поскольку была определена принудительно не как число типа double, а как число типа 64-bit integer с номером для поиска 18 (см. приложение С). Таким образом, чтобы проверить, какие документы в поле **age** имеют 64-битное целое число, следует выполнить команду

```
db.personnel.find( {age: { $type : 18 } } );
```

Результат ее выполнения представлен на рис. 8.28.

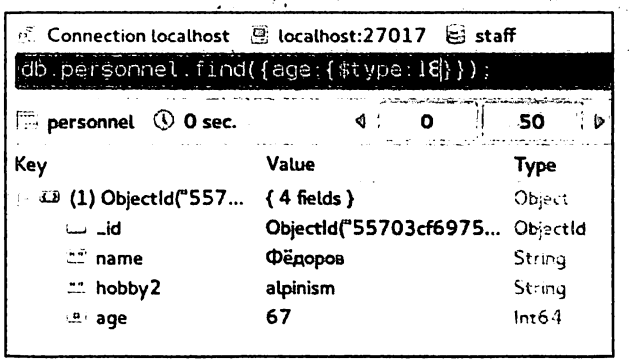


Рис. 8.28. Выбор документов, у которых в поле **name** тип данных число типа 64-bit integer

В качестве самостоятельного упражнения читатель может внести различные типы данных в конкретное поле документов и осуществить выборку по определенному типу данных.

Необходимо также обратить внимание на то, что MongoDB не поддерживает объединение коллекций, т. е. оператор **JOIN** в привычном смысле не функционирует. Для возможности масштабирования MongoDB данные хранятся в виде коллекций *документов* с соответствующими данными в документах, таким образом, чтобы устранить необходимость соединений с другими коллекциями. Тем не менее в некоторых случаях имеет смысл хранить соответствующую информацию в отдельных документах, как правило, в разных коллекциях или базах данных.

В MongoDB применяется один из двух методов:

- сохранение поля **_id** одного документа в другом документе в качестве ссылки. Тогда приложение может выполнять запрос данных из второго документа. Эти ссылки являются простыми, и их достаточно для большинства случаев;
- ***DBRefs*** являются ссылками из одного документа в другой с использованием значения поля первого документа **_id**, названия коллекции и при необходимости имени базы данных. Это позволяет использовать документы, расположенные в нескольких коллекциях.

Рассмотрим пример, каким образом работает первый метод. Для этого создадим коллекцию **education** для хранения данных об образовательных учреждениях (рис. 8.29). Далее поле **_id** (см. выше) из коллекции **education** сохраним в качестве ссылки в некотором поле

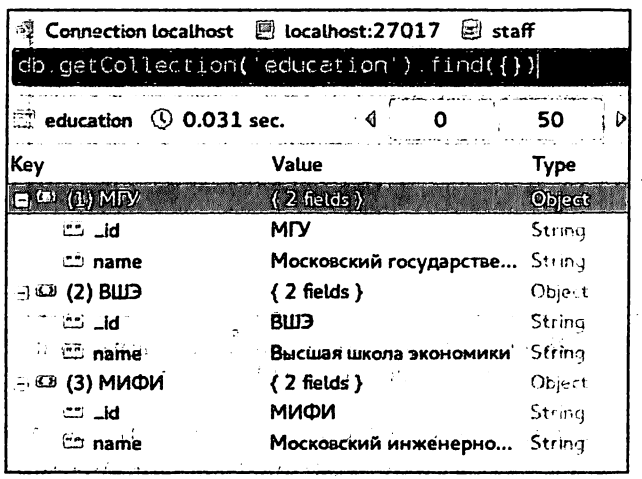


Рис. 8.29. Создание коллекции **education**

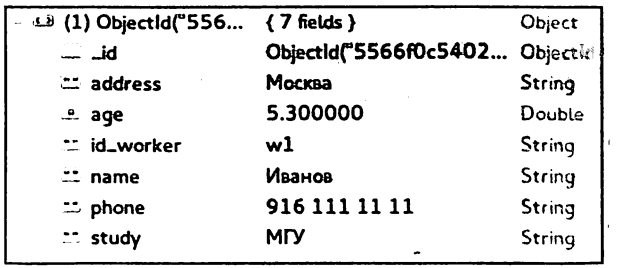


Рис. 8.30. В поле **study** заносится **_id** коллекции **education**

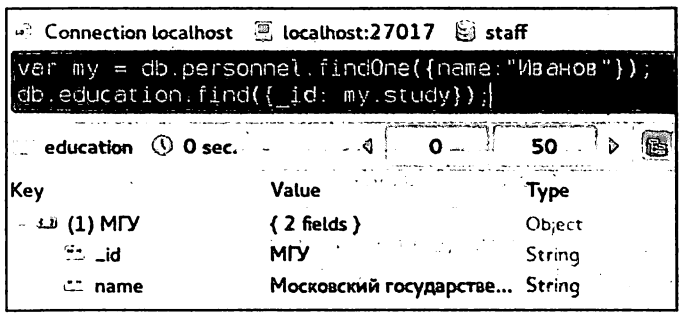


Рис. 8.31. Ссылка на коллекцию **education**

(например, в документе с именем «Иванов» создадим поле **study**, см. (рис. 8.30)). После этого при осуществлении поиска по имени «Иванов» и полю **study** осуществляется ссылка на коллекцию **education** (рис. 8.31):

```
var my = db.personnel.findOne({name:"Иванов"});
db.education.find({_id: my.study});
```

Обратите внимание, что для поиска была использована команда **findOne()**, которая непосредственно возвращает первый найденный документ и позволяет использовать точку (т. е. **my.study**) для доступа к полю (в отличие от команды **find()**, которая возвращает множество найденных документов).

Рассмотрим второй метод организации ссылок. Для разнообразия будем использовать метод **save()**, который в данном случае практически полностью аналогичен методу **insert()** (детали см. в соответствующей документации). В качестве упражнения читатель может использовать метод **insert()** при создании собственных ссылок (результат бу-

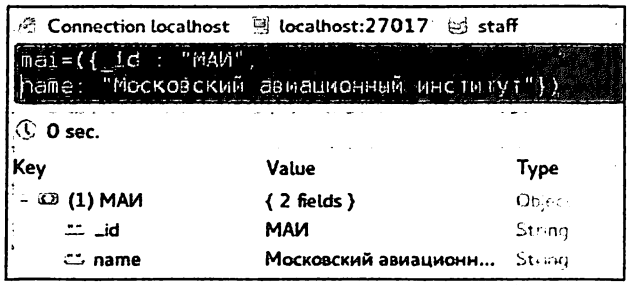


Рис. 8.32. Создание нового документа

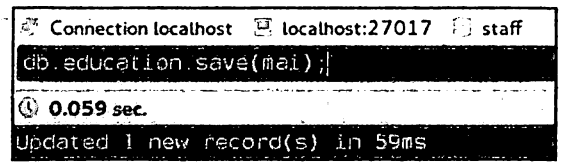


Рис. 8.33. Сохранение нового документа

дет точно таким же). Сначала необходимо добавить новый документ в коллекцию **education** (рис. 8.32 и 8.33):

```
mai={{_id : "МАИ", name: "Московский авиационный институт"});
db.education.save(mai);
```

После этого коллекция **education** имеет следующий вид (рис. 8.34).

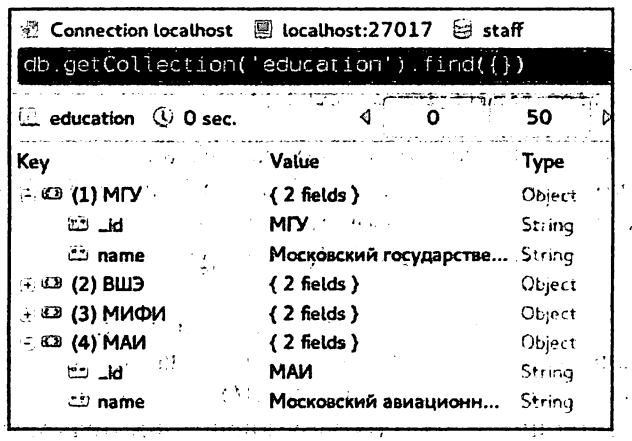


Рис. 8.34. Коллекция **education**

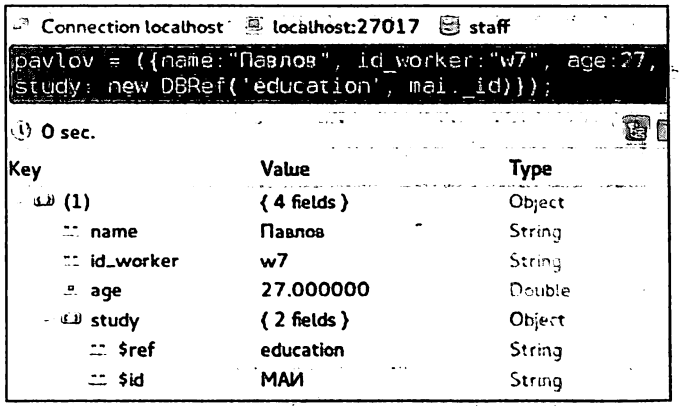


Рис. 8.35. Внесение данных о новом сотруднике

Внесем данные о новом сотруднике и сохраним их (рис. 8.35):

```
pavlov = ({name:"Павлов", id_worker:"w7", age:27, study: new
DBRef('education', mai._id)});
db.personnel.save(pavlov);
```

Проверим, что данные внесены (рис. 8.36).

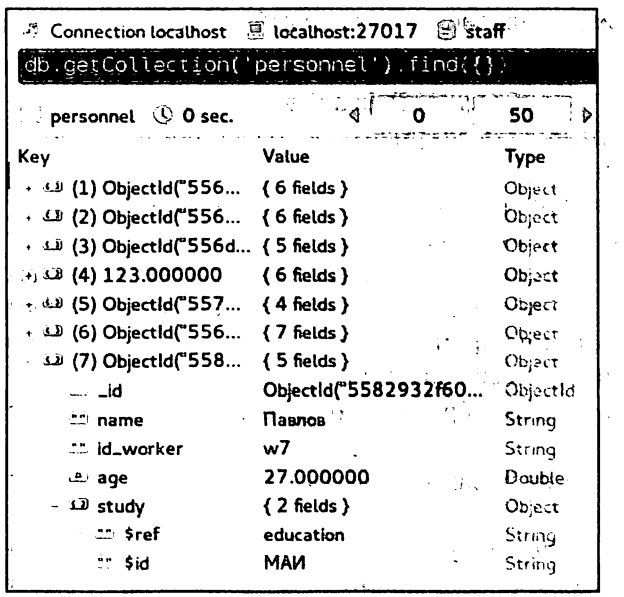


Рис. 8.36. Данные о новом сотруднике внесены

Проверяем возможность доступа по ссылке (рис. 8.37):

```
db.education.findOne({_id: pavlov.study.$id});
```

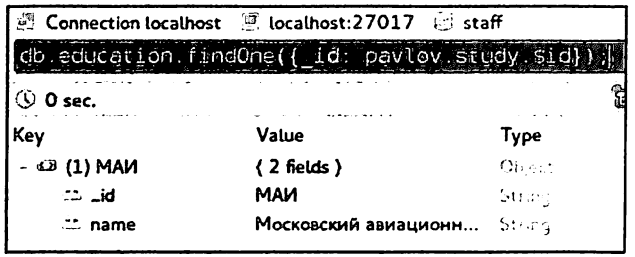


Рис. 8.37. Доступ к другой коллекции по ссылке

На основе данных принципов работы со ссылками имеется возможность грамотно масштабировать базы данных и использовать различные коллекции, если это необходимо.

Учащимся предлагается выполнить практическую работу, основанную на изученном теоретическом материале. Выполнение практической работы предполагает освоение навыков выборки данных из коллекций MongoDB.

Порядок выполнения практической работы

1. Ознакомиться с описанием.
2. Запустить оболочку **Robomongo**.
3. Отсортировать документы в коллекции по полю, определенному в задании.
4. Осуществить выборку по условию, определенному в задании, составив регулярное выражение.
5. Сгруппировать документы по некоторому полю, ввести условие группировки в соответствии с заданием. Сравнить результаты.
6. Вывести на экран определенное число документов, начиная с определенного номера документа.
7. Задать явным образом тип данных поля (полей) в соответствии с заданием.
8. Вывести на экран документы, тип данных полей которых соответствует типу данных из п.7.
9. Создать еще одну коллекцию в соответствии с заданием.
10. Одним из рассмотренных выше способом создать ссылки на вновь созданную коллекцию.

11. Сохранить базу данных и выйти из программы.
12. Подготовить отчет по практической работе, содержащий результаты выполнения основных запросов при работе с документами и создания ссылок на вновь созданную коллекцию в оболочке **Robomongo**.

Контрольные вопросы

1. При помощи какой команды осуществляется сортировка записей по полю?
2. Какие селекторы запросов вы знаете? Перечислите не менее пяти.
3. Каким образом осуществляется группировка по полю? Как задается условие группировки?
4. Можно ли ограничить число отображаемых на экране документов?
5. Для чего служит команда **skip()**?
6. Как задать принудительно тип данных?
7. Можно ли выделить поля, содержащие определенный тип данных?
8. Расскажите об особенностях команды **findOne()**.
9. Расскажите, как происходит объединение коллекций при сохранении поля **_id** одного документа в другом документе в качестве ссылки?
10. Какой еще способ объединения коллекций вы знаете?

Лабораторная работа 9
ОСНОВЫ АДМИНИСТРИРОВАНИЯ СУБД MONGODB

Цель: приобретение основных навыков администрирования **MongoDB**: создание пользователей, ролей, сохранение и восстановление баз данных.

Администрирование сервера **mongod** является достаточно сложной задачей, поэтому необходимо обладать базовыми навыками основ администрирования для эффективной работы с СУБД. Следует помнить, что для выполнения администрирования пользователь должен иметь соответствующие права.

Приведем небольшой список команд, которые используются при администрировании (часть из них уже была описана выше).

Команда **help** — посмотреть справку. При ее запуске ответ сервера выглядит следующим образом:

```
db.help()          help on db methods
db.mycoll.help()   help on collection methods
sh.help()          sharding helpers
rs.help()          replica set helpers
help admin         administrative help
help connect       connecting to a db help
help keys          key shortcuts
help misc          misc things to know
help mr            mapreduce
show dbs           show database names
show collections   show collections in current database
show users         show users in current database
show profile       show most recent system.profile entries with
time >= 1ms
show logs          show the accessible logger names
show log [name]    prints out the last segment of log in memory,
'global' is default
use <db_name>      set current database
db.foo.find()      list objects in collection foo
```



```
db.foo.find( { a : 1 } ) list objects in foo where a == 1
it    result of the last line evaluated; use to further iterate
DBQuery.shellBatchSize = x set default number of items to
display on shell
exit               quit the mongo shell
```

Команда **db.help()** позволяет получить справку по **db methods**, а **db.commandHelp(«команда»)** — справку по определенной команде.

Как было сказано выше, команда **show dbs** позволяет получить список БД на сервере, **show tables / show collections** — список таблиц/коллекций текущей базы, **show users** — список пользователей в текущей БД.

Для того чтобы сделать базу данных текущей, используется команда **use <имя базы данных>**.

Команда **db.version()** позволяет вывести версию программы.

Для получения данных по текущему состоянию сервера используется команда **db.serverStatus()**. Пример (здесь ответ сервера ввиду его обширности приводится не полностью):

```
db.serverStatus()
{
  "host" : "mycomputer.localdomain",
  "version" : "2.6.11",
  "process" : "mongod",
  "pid" : NumberLong(1283),
  "uptime" : 5816,
  "uptimeMillis" : NumberLong(5816717),
  "uptimeEstimate" : 5771,
  "localTime" : ISODate("2015-11-17T11:52:46.182Z")
....
```

Команда **db.shutdownServer()** позволяет завершить корректно и безопасно соединение **mongod** или **mongos** (последний термин относится к процессу — маршрутизатору и используется при организации шардинга).

Команда **db.runCommand(*command*)** является предпочтительным методом для выполнения определенных команд базы данных, поскольку обеспечивает единый интерфейс между оболочкой.

Данная команда принимает следующие аргументы (табл. 9.1).

Команда **buildInfo** базы данных возвращает документ, содержащий обзор параметров, используемых в текущем соединении **mongod**. Вызов команды:

```
db.runCommand({ buildInfo: 1 })
```

Таблица 9.1. **Аргументы команды db.runCommand()**

Параметр	Тип данных	Описание
command	document or string	Запускает команду в контексте текущей базы данных

Отклик (ввиду большого объема информации приводится не полностью):

```
{
 "version" : "2.6.11",
 "gitVersion" : "nogitversion",
 "OpenSSLVersion" : "OpenSSL 1.0.1k-fips 8 Jan 2015",
 "sysInfo" : "Linux buildhw-09.phx2.fedoraproject.org 4.
...
```

Команда **db.stats()** предназначена для получения статистики по текущей базе данных. Для иллюстрации данной команды перейдем в базу данных **test**:

```
use test
switched to db test
```

и рассмотрим пример:

```
db.stats()
{
  "db" : "test",
  "collections" : 3,
  "objects" : 5,
  "avgObjSize" : 53.6,
  "dataSize" : 268,
  "storageSize" : 16384,
  "numExtents" : 3,
  "indexes" : 1,
  "indexSize" : 8176,
  "fileSize" : 201326592,
  "nsSizeMB" : 16,
  "dataFileVersion" : {
   "major" : 4,
   "minor" : 5
  },
  "extentFreeList" : {
   "num" : 3,
   "totalSize" : 49152
  },
  "ok" : 1
}
```


Пользователи

Важной частью администрирования является администрирование прав доступа пользователей. Для аутентификации клиента в MongoDB необходимо добавить соответствующего пользователя в MongoDB. Чтобы добавить пользователя, в MongoDB используется метод **db.createUser()**. При добавлении пользователя ему могут быть назначены роли. Заметим, что начало работы с базой происходит в режиме администратора, который имеет право выполнять всевозможные действия с другими пользователями: создавать нового, изменять пароль, назначать права доступа, а также удалять пользователя.

При добавлении пользователя в общем случае он создается для конкретной базы данных. Имя и аутентификация в базе данных пользователя служат в качестве уникального идентификатора для этого пользователя. Таким образом, если два пользователя имеют одинаковое имя, но создаются в различных базах данных, то они представляют собой двух отдельных пользователей. Если же одному пользователю необходимо обладать правами на выполнение операций в нескольких базах данных, то следует создать одного пользователя с определенными ролями для этих баз данных вместо создания нескольких пользователей в разных базах данных.

Программный код для создания нового пользователя имеет вид:

```
{ user: "<name>",
 pwd: "<cleartext password>",
 customData: { <any information> },
 roles: [
  { role: "<role>", db: "<database>" } | "<role>",
  ...
 ]
}
```

Поля имеют следующие значения (табл. 9.2).

В поле **roles** (роли) могут быть внесены как специфицированные в MongoDB роли, так и роли, созданные пользователем.

Чтобы указать роль, которая существует в той же базе данных, где выполнен метод **db.createUser()**, можно просто указать имя роли, например: «**readWrite**».

Можно указать роль в связке с базой данных:

```
{ role: "<role>", db: "<database>" }
```

Таблица 9.2. **Значение полей при создании нового пользователя**

Поле	Тип данных	Описание
user	string	Имя пользователя
pwd	string	Пароль пользователя
customData	document	Необязательное поле, содержащее произвольную информацию, которую администратор хочет занести для данного конкретного пользователя. Например, это может быть полное имя пользователя или идентификатор сотрудника
roles	array	Список ролей пользователя. Может быть пустой массив [ ], если создается пользователь без ролей

Для указания роли, привязанной к разным базам данных, следует указывать роль и базу данных.

В качестве примера осуществим добавление нового пользователя в базу данных **test**. Далее будем работать через консоль, поскольку в зависимости от версии MongoDB оболочка Robomongo поддерживает не все команды. Откроем консоль и наберем в командной строке:

```
db.createUser( { "user" : "user1",
 "pwd": "password1",
 "roles" : [ { role: "readWrite", db: "test" }] }
... )
```

Если пользователь будет внесен успешно, то появится сообщение:

```
Successfully added user: {
 "user" : "user1",
 "roles" : [
  {   "role" : "readWrite",
      "db" : "test"
  }
 ]
}
```

Для проверки, какие пользователи существуют в базе данных, выполним команду:

```
show users
```

Ответ сервера:

```
{{
 "_id" : "test.user1",
```



```
 "user" : "user1",
 "db" : "test",
 "roles" : [
  {
    "role" : "readWrite",
    "db" : "test"
  }
 ]
}
```

Изменение (update) профиля пользователя производится в той базе данных, в которой выполняется метод **db.updateUser(username, update, writeConcern)**. При этом в заменяемых полях предыдущие значения заменяются полностью, в том числе и массив ролей пользователя. Если необходимо добавить или удалить роли без замены всего пользователя, следует использовать методы **db.grantRolesToUser()** или **db.revokeRolesFromUser ()**.

Метод **db.updateUser ()** использует следующий синтаксис:

```
db.updateUser(
 "<username>",
 {
  customData : { <any information> },
  roles : [
     { role: "<role>", db: "<database>" } | "<role>",
     ...
     ],
  pwd: "<cleartext password>"
 },
 writeConcern: { <write concern> }
)
```

Метод **db.updateUser ()** имеет следующие аргументы (табл. 9.3).

Таблица 9.3. **Аргументы метода db.updateUser ()**

Параметр	Тип данных	Описание
username	string	Имя пользователя, для которого производится обновление
update	document	Документ, содержащий данные для замены. Эти данные полностью заменяют имеющиеся данные
writeConcern	document	Необязательный параметр, контроль операции записи

В обновляемом документе определяются изменяемые поля для замены их на новые значения. Все поля в обновляемом документе не являются обязательными, но обновление должно включать в себя, по меньшей мере, одно поле.

Обновляемый документ имеет следующие поля (табл. 9.4).

Таблица 9.4. **Поля обновляемого документа**

Поле	Тип данных	Описание
customData	document	Необязательное поле, содержащее произвольную информацию
roles	array	Необязательное поле. Список ролей пользователя. Заменяет полностью предыдущий массив ролей пользователя
pwd	string	Необязательное поле. Пароль пользователя

В качестве примера рассмотрим изменения пароля пользователя: заменим **password1** на **password12.**

```
db.updateUser(
... "user1",
... {
... pwd: "password11"
... }
... )
```

Для изменения пароля пользователя можно использовать специальный метод (запускается в базе данных, где определен пользователь) **db.changeUserPassword(*username*, *password*).**

В качестве примера вернем значение пароля пользователя **password1**:

```
db.changeUserPassword("user1", "password1")
```

Для проверки правильности выполненных действий следует использовать команду

```
show users
```

Для удаления всех пользователей из текущей базы данных используется метод **db.dropAllUsers(*writeConcern*),** для удаления конкретного пользователя из текущей базы данных используется метод **db.dropUser(*username*, *writeConcern*).** Параметр **username** — имя пользователя в соответствующей базе данных. Обратите внимание, что метод удаления пользователя из базы данных **db.removeUser(*username*)** является устаревшим и может не поддерживаться в более поздних версиях MongoDB.

Пример:

```
db.dropUser("user1")
```

Ответ сервера:

```
True
```

Проверим наличие пользователя:

```
show users
```

В качестве ответа не получим ни одного пользователя:

```
>
```

Для получения информации о пользователе используется метод **db.getUser(*username*).** Метод следует запускать в той базе данных, где этот пользователь существует. Метод принимает один параметр: **username** (строка) — имя пользователя, о котором должна быть получена информация. Для получения информации обо всех пользователях базы данных следует запустить метод **db.getUsers():**

```
db.getUser("user1")
{
  "_id" : "test.user1",
  "user" : "user1",
  "db" : "test",
  "roles" : [
   {
      "role" : "readWrite",
      "db" : "test"
   }
 ]
}
```

Роли

Встроенные роли

MongoDB предоставляет доступ к данным и методам на основе авторизации пользователя с указанными ролями, которые обеспечи-

вают различные права доступа (привилегии). В MongoDB имеются как встроенные роли, так и возможность дополнительно создавать пользовательские роли.

Каждая база данных включает в себя следующие роли **клиента**: **read** (чтение) и **readWrite** (чтение и запись). Последняя роль предполагает возможность в том числе создавать, изменять и удалять коллекции.

Роль **администратора базы** данных **dbAdmin** предполагает стандартные права доступа администратора к той базе данных, где эта роль определена. Роль администратора включает в себя также роли для резервного копирования и восстановления данных: **backup** (резервное копирование, эта роль обеспечивает минимальные привилегии, необходимые для резервного копирования данных) и **restore** (восстановить, эта роль обеспечивает привилегии для восстановления данных из резервных копий).

Роль **UserAdmin** предоставляет возможность создавать и изменять роли и пользователей в текущей базе данных.

dbOwner — владелец базы данных может выполнять любые административные действия в базе данных. Эта роль сочетает в себе привилегии, предоставленные роли **ReadWrite**, **dbAdmin** и **UserAdmin**.

При работе на кластере существуют роли **clusterAdmin**, **clusterManager** и **clusterMonitor**.

Роль **hostManager** предоставляет возможность мониторинга и управления серверами.

Роли для всех баз данных конкретного сервера **mongod** (за исключением кластеров):

- **readAnyDatabase** (чтение любой базы данных);
- **readWriteAnyDatabase** (чтение и запись);
- **userAdminAnyDatabase** (администрирование пользователей);
- **dbAdminAnyDatabase** (администрирование баз данных).

Роль **root** обеспечивает доступ к операциям и всем ресурсам как роли: **readWriteAnyDatabase**, **dbAdminAnyDatabase**, **userAdminAnyDatabase** и **clusterAdmin**, вместе взятые.

Для создания роли, определяемой пользователем, используется метод **db.createRole(*role*, *writeConcern*)**.

Привилегии для роли можно указать явно.

Роль относится к той базе данных, для которой выполняется метод.

Метод принимает следующие аргументы (табл. 9.5).

Таблица 9.5. **Аргументы метода db.createRole()**

Параметр	Тип данных	Описание
roles	document	Документ содержит имя роли и ее определение
writeConcern	document	Необязательный параметр, контроль операции записи

Документ «**roles**» имеет следующий вид:

```
{
 role: "<name>",
 privileges: [
  { resource: { <resource> }, actions: [ "<action>", ... ] },
  ...
 ],
 roles: [
  { role: "<role>", db: "<database>" } | "<role>",
  ...
 ]
}
```

Описание полей документа «**roles**» представлено в табл. 9.6.

Таблица 9.6. **Поля документа «roles»**

Поле	Тип данных	Описание
role	string	Имя новой роли
privileges	array	Права (привилегии) роли. Привилегия состоит из ресурса (базы данных) и разрешенных действий. Поле привилегий должно существовать. Если не указываются привилегии, следует использовать пустой массив
roles	array	Массив ролей, из которых эта роль наследует привилегии. Поле должно существовать. Если роли не наследуются, следует использовать пустой массив

Рассмотрим пример создания роли **role1**, для которой предусмотрены привилегии для всех баз данных и коллекций выполнять поиск, а также встроенную роль чтение—запись для базы данных **test**.

Создание роли:

```
> db.createRole(
... {
...   role: "role1",
```

```
...  privileges: [
...   {
...     resource: { db: "", collection: "" },
...     actions: [ "find" ]
...   }
...  ],
...  roles:
...  [
...   { role: "readWrite", db: "test" }
...  ]
... }
... )
```

Ответ сервера:

```
{
 "role" : "role1",
 "privileges" : [
  {
   "resource" : {
    "db" : "",
    "collection" : ""
   },
   "actions" : [
    "find"
   ]
  }
],
"roles" : [
 {
  "role" : "readWrite",
  "db" : "test"
 }
]
}
```

Для изменения ролей пользователя используется метод, запущенный в той базе данных, для которой эти роли определены **db.updateRole(*rolename*, *update*, *writeConcern*)**. Заметим, что при обновлении массива привилегий или ролей значения предыдущего массива полностью заменяются.

Синтаксис метода:

```
db.updateRole(
 "<rolename>",
 {
  privileges:
   [
```



```
    { resource: { <resource> }, actions: [ "<action>", ... ] },
    ...
   ],
  roles:
   [
     { role: "<role>", db: "<database>" } | "<role>",
     ...
   ]
  },
  { <writeConcern> }
)
```

Метод принимает аргументы (табл. 9.7).

Таблица 9.7. **Аргументы метода db.updateRole()**

Параметр	Тип данных	Описание
rolename	string	Название определяемой пользователем роли, которое планируется изменить
update	document	Документ, содержащий данные, которыми будут заменены текущие
writeConcern	document	Необязательный параметр, контроль операции записи

Каждое поле в документе обновления (**update**) не является обязательным, но документ должен содержать, по крайней мере, одно поле. Документ обновления имеет следующие поля (табл. 9.8).

Таблица 9.8. **Аргументы документа обновления**

Поле	Тип данных	Описание
privileges	array	Необязательное поле. Обновление в массиве привилегий отменяет значения предыдущего массива
roles	array	Необязательное поле. Список ролей, с которой эта роль наследует привилегии. Обновление массива ролей отменяет значения предыдущего массива

Изменим роль, добавив в привилегии не только поиск, но и изменение и вставку данных:

```
db.runCommand(
... {
...  updateRole: "role1",
...  privileges: [
...   {
```

```
...    resource: { db: "", collection: "" },
...    actions: [ "find" , "update", "insert"]
...   }
...  ],
...  roles:
...  [
...   { role: "readWrite", db: "test" }
...  ]
... }
... )
```

Ответ сервера:

```
{ "ok" : 1 }
```

Для удаления ролей, определенных пользователем в базе данных, в которой они определены, используется метод: **db.dropRole(*rolename*, *writeConcern*)**.

Метод принимает следующие аргументы (табл. 9.9).

Таблица 9.9. **Аргументы метода db.dropRole()**

Параметр	Тип данных	Описание
rolename	string	Название роли, определенной пользователем, которую требуется удалить из базы данных
writeConcern	document	Необязательный параметр, контроль операции записи

Пример:

```
db.dropRole("role2")
```

Ответ сервера:

```
true
```

Для удаления всех ролей используется метод **db.dropAllRoles (*writeConcern*)**, который удалит все определенные пользователем роли из базы данных.

Для добавления привилегий в определенные пользователем роли служит метод **db.grantPrivilegesToRole(*rolename*, *privileges*, *writeConcern*)**, который имеет следующий синтаксис:

```
db.grantPrivilegesToRole(
 "< rolename >",
 [
  { resource: { <resource> }, actions: [ "<action>", ... ] },
  ...
 ],
 { < writeConcern > }
)
```


Аргументы метода следующие (табл. 9.10).

Таблица 9.10. **Аргументы метода db.grantPrivilegesToRole ()**

Параметр	Тип данных	Описание
rolename	string	Название роли, которой добавляются привилегии
privileges	array	Список привилегий, добавляемых роли
writeConcern	document	Необязательный параметр, контроль операции записи

Рассмотрим пример, в котором добавляется возможность для роли **role1** удалять коллекции в базе данных **test**.

```
> db.grantPrivilegesToRole(
... "role1",
... [
...  {
...    resource: { db: "test", collection: "" },
...    actions: [ "remove" ]
...  }
... ]
... )
```

Если необходимо назначить пользователю дополнительные роли, то вызывается метод **db.grantRolesToUser()**, который имеет следующий синтаксис:

```
db.grantRolesToUser( "<username>", [ <roles> ], {
<writeConcern> } )
```

Метод принимает следующие аргументы (табл. 9.11).

В поле **roles** (роли) могут быть внесены как специфицированные в MongoDB роли, так и роли, созданные пользователем.

Метод **db.getRole(*rolename*, *showPrivileges*)** возвращает список ролей, из которых данная роль наследует привилегии.

Таблица 9.11. **Аргументы метода db.grantRolesToUser ()**

Параметр	Тип данных	Описание
user	string	Имя пользователя, которому добавляются роли
roles	array	Массив ролей, которые назначаются пользователю
writeConcern	document	Необязательный параметр, контроль операции записи

Данный метод принимает следующие аргументы (табл. 9.12).

Таблица 9.12. **Аргументы метода db.getRole ()**

Параметр	Тип данных	Описание
rolename	string	Название роли
showPrivileges	document	Необязательный параметр. Возвращает привилегии роли, если имеет значение истина {showPrivileges: true}

Для удаления одной или нескольких ролей пользователя в базе данных используется метод **db.revokeRolesFromUser()** (или метод **db.revokePrivilegesFromRole(*rolename*, *privileges*, *writeConcern*)**).

Метод имеет следующий синтаксис:

```
db.revokeRolesFromUser( "<username>", [ <roles> ], {
<writeConcern> } )
```

Метод принимает следующие аргументы (табл. 9.13).

Таблица 9.13. **Аргументы метода db.revokeRolesFromUser ()**

Параметр	Тип данных	Описание
user	string	Имя пользователя, у которого удаляются роли
roles	arrayв	Список ролей, которые удаляются
writeConcern	document	Необязательный параметр, контроль операции записи

В поле **roles** (роли) могут быть внесены как специфицированные в MongoDB роли, так и роли, созданные пользователем.

Инструменты MongoDB для резервного копирования и восстановления

Заметим, что для создания резервной копии необходимо, чтобы пользователь обладал соответствующими правами.

Утилита **mongodump** позволяет осуществить резервное копирование данных при подключении к активному серверу **mongod** или **mongos**. Утилита может создать резервную копию для всего сервера, для базы данных, для коллекции либо только для части коллекции (последнее — при использовании соответствующего запроса). Следует обратить внимание, что, если в директории, куда производится запись, имеются файлы с аналогичным названием, они будут перезаписаны, а предыдущая информация утеряна.

При запуске команды **mongodump** без каких-либо аргументов команда подключается к соединению MongoDB в локальной системе (например, 127.0.0.1 или localhost) на порт 27017 и создает резервную копию базы данных с именем **dump/** в текущем каталоге. Также можно указать **--host** и **--port** соединения с MongoDB. Например:

```
mongodump --host mongodb.example.net --port 27017
```

Чтобы указать другой каталог вывода, можно использовать опцию **--out** или **--o**:

```
mongodump --out /data/backup/
```

Например, следующая команда отправит дамп базы данных **test** в указанную директорию **home/testdb**:

```
mongodump -db test -o /home/testdb
```

Чтобы ограничить объем данных, включенных в дамп базы данных, можно указать **--db** и **—collection** в качестве параметров для **mongodump**. Например:

```
mongodump --collection myCollection --db test
```

Эта операция создает дамп с именем **myCollection** из базы данных **test** в подкаталог **dump/** текущей рабочей директории.

Параметры **mongodump --host** и **--port** позволяют подключаться для резервного копирования с удаленного хоста:

```
mongodump --host mongodb1.example.net --port 3017 --username
user --password pass --out /opt/backup/mongodump-2015-01-01
```

Восстановление базы данных с mongorestore

Заметим, что для восстановления базы данных необходимо, чтобы пользователь обладал соответствующими правами.

Утилита **mongorestore** восстанавливает резервную копию, созданную **mongodump**. По умолчанию **mongorestore** ищет резервную копию базы данных в каталоге **dump/**. Утилита восстанавливает данные **mongorestore** при непосредственном подключении к **mongod** или **mongos**. Утилита **mongorestore** может восстановить резервную копию базы данных либо полностью, либо ее подмножество. Общий вид команды:

```
mongorestore --port <port number> <path to the backup>
```

По умолчанию **mongorestore** работает на **localhost** (например, 127.0.0.1) и имеет порт по умолчанию (27017). Например, команда

```
mongorestore -- db test -o /home/testdb
```

восстанавливает базу данных **test** из дампа из директории **home/testdb** на **localhost**.

Если необходимо восстановить базу данных с удаленного хоста или порта, это можно сделать, явно указав параметры **--host** и **--port**.

```
mongorestore --host mongodb1.example.net --port 3017
--username user --password pass
/opt/backup/mongodump-2015-01-01
```

Для экспорта-импорта данных иных форматов (CSV, JSON и пр.) следует использовать операции **mongoexport** и **mongoimport**, которые не рассматриваются ввиду их сложности.

Учащимся предлагается выполнить практическую работу, основанную на изученном теоретическом материале. Выполнение практической работы предполагает освоение основных навыков администрирования MongoDB, а именно создание пользователей, ролей, сохранение и восстановление баз данных.

Порядок выполнения практической работы

1. Ознакомиться с описанием.
2. Запустить сервер **mongod** в режиме администрирования.
3. Проверить существующие настройки.

4. Получить у преподавателя задание с именем нового пользователя и перечнем его прав и ролей, включая роли, созданные пользователем.
5. Получить у преподавателя задание с именем баз данных для выполнения экспорта и импорта.
6. Создать нового пользователя с соответствующими правами.
7. Создать роль в соответствии с заданием. Добавить созданную роль в список ролей созданного пользователя.
8. Выбрать базу данных для экспорта. Экспортировать ее в соответствии с заданием.
9. В соответствии с заданием импортировать базу данных. Проверить, что база данных импортирована.
10. Сохранить результаты выполнения работы и выйти из программы.

Контрольные вопросы

1. Для чего предназначен режим администрирования?
2. Расскажите, каким образом можно посмотреть статус сервера? Информацию о существующих на нем базах данных?
3. Для чего предназначена команда **help**?
4. Из каких шагов состоит процесс создания нового пользователя?
5. Каким образом назначаются пользователю привилегии?
6. Какие имеющиеся (встроенные) роли могут быть назначены пользователю (назовите не менее трех)?
7. Из каких шагов состоит процесс создания роли, определяемой пользователем?
8. Каким образом осуществляется резервное копирование? Какие настройки необходимо выполнить?
9. Для каких целей используется операция **mongorestore**?
10. Какие параметры необходимо указать для сохранения/извлечения данных с удаленного хоста?

Лабораторная работа 10
РЕПЛИКАЦИЯ И ШАРДИНГ В СУБД MONGODB

Цель: изучение способов повышения надежности функционирования СУБД MongoDB при помощи репликации и приобретение навыков горизонтального масштабирования данныхв СУБД MongoDB при помощи шардинга.

Репликация и шардинг являются важнейшими технологиями СУБД MongoDB, поскольку позволяют организовать распределенные вычисления.

Репликация

Под репликацией понимается размещение и обслуживание серверов базы данных на нескольких физических узлах. Цель репликации — обеспечить избыточность и обработку отказа (например, для восстановления после сбоя аппаратного обеспечения) и увеличить доступность и балансировку данных (например, для распределенных приложений).

В **MongoDB** имеется два типа репликации: первый тип — «главный—подчиненный» (**Master Slave Replication**), второй тип — «наборы реплик» (**Replica Sets**). Последний является развитием механизма репликации **Master Slave Replication** и сохраняет основные ее принципы работы. И в том, и в другом случае имеется первичный узел (**Primary**), который выполняет все операции записи, и вторичные узлы (**Secondary**), которые считывают описания операций, выполненных на первичном узле, и применяют их. Однако использование наборов реплик предпочтительнее ввиду наличия более широких возможностей, например автоматического восстановления (в частности, в случае отказа первичного узла имеется механизм автоматического назначения первичным узлом одного из вторичных).

В MongoDB репликация асинхронная, поскольку синхронизация данных происходит с некоторой задержкой, а не в момент принятия изменений. На первичном узле создается лог (журнал) операций (**oplog**), который посылается на вторичные узлы для обновления. Следует помнить, что поскольку лог имеет конечные размеры, то при переполнении более старые операции будут заменяться новыми.

Рассмотрим настройку и конфигурирование набора реплик. В минимальной желательной конфигурации имеется три узла (**members**), два из которых серверы **mongod**, любой из них может стать первичным (второй соответственно вторичным). Третий узел, который называется арбитр (**Arbiter**), не содержит копию данных. Его функциональность ограничивается хранением конфигурации **Replica Sets**, а в случае отказа помогает выбрать новый первичный узел. При этом участники отправляют друг другу каждые две секунды тактовые сигналы (**heartbeats**), и, если в течение 10 с от узла нет ответа, остальные члены набора считают его недоступным — т. е. произошел отказ.

Из сказанного следует, что нежелательно запускать арбитр на одной машине с другим членом (первичным или вторичным узлом) того же набора реплик.

Пример создания репликации. Создадим на одном компьютере конфигурацию **Replica Sets**, состоящую из трех узлов: первичного, вторичного и арбитра. Для этого первоначально создадим основную директорию и три поддиректории, в которых на диске будут храниться данные. В MongoDB по умолчанию директорией для хранения данных репликации является директория **data**. В директории **data** создадим поддиректорию **myrepl** для хранения создаваемой репликации, а в ней поддиректории **node1**, **node2** и **arbiter** для размещения данных узла 1, узла 2 и арбитра соответственно. Директории можно создать, работая в **mc (Midnight Commander)** или **Krusader**. Если работа идет в командной строке, то достаточно набрать команды (напоминаем, что по умолчанию поиск данных будет осуществляться в директории **data/** (рис. 10.1):

```
mkdir /data/myrepl/node1
mkdir /data/myrepl/node2
mkdir /data/myrepl/arbiter
```

После создания места для хранения данных можно приступить к репликации. Для запуска трех процессов откроем три консоли и за-

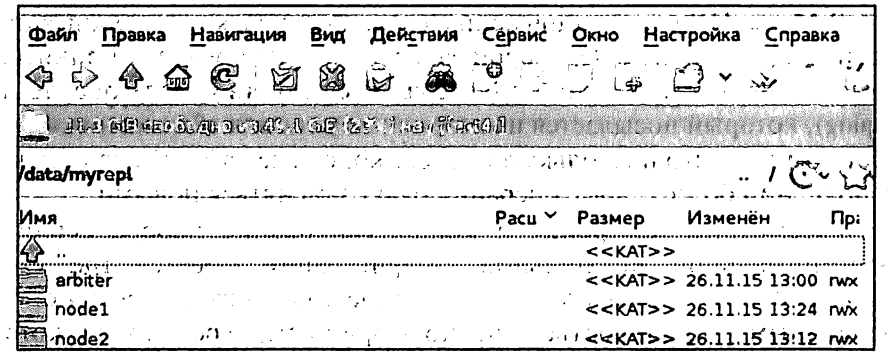


Рис. 10.1. Создание поддиректорий для репликации

пустим свой **mongod** в каждой из них. В первой консоли запустим процесс создания первого узла репликации, которую назовем **myrs**, через порт 30000 (порт можно выбрать произвольно из числа разрешенных (см. документацию)):

```
mongod --replSet myrs --dbpath /data/myrepl/node1 --port 30000
```

Через некоторое время (не сразу, поскольку процесс должен запуститься) в качестве ответа получим (ввиду большого количества информации приводим только начало ответа):

```
2015-11-25T14:50:35.522+0300 [initandlisten] connection
accepted from 127.0.0.1:48318 #10 (2 connections now open)...
```

Во второй консоли запустим процесс создания второго узла репликации через порт 30001:

```
mongod --replSet myrs --dbpath /data/myrepl/node2 --port 30001
```

В третьей консоли запустим процесс создания арбитра репликации через порт 30002:

```
mongod --replSet myrs --dbpath /data/myrepl/arbiter --port
30002
```

Запуск может занять некоторое время.

Для конфигурации реплик откроем отдельное окно. В процессе работы для удобства будем держать открытыми четыре окна: три для запущенных реплик, в которых постоянно будет отображаться информация, отражающая происходящие процессы на узле, и одно для

ввода команд (рис. 10.2). Заметим, что окна, которые не используются в данный момент, можно перевести в фоновый режим (как это сделать — см. документацию). Но при выполнении учебного примера лучше держать все окна открытыми.

Рис. 10.2. Открытые окна процессов

К узлу обращаемся по локальному имени компьютера (в нашем случае **my_comp**), поскольку все процессы запущены локально:

```
mongo --port 30000 --host my_comp.localdomain
```

В качестве ответа получим подтверждение соединения:

```
$ mongo --port 30000 --host my_comp.localdomain
MongoDB shell version: 2.6.11
connecting to: my_comp.localdomain:30000/test
```

После подключения запустим команду, которая инициирует набор реплик:

```
rs.initiate()
```

Ответ показывает, что получен набор реплик с одним членом:

```
{
 "info2" : "no configuration explicitly specified -- making one",
 "me" : " my_comp.localdomain:30000",
 "info" : "Config now saved locally. Should come online in
 about a minute.",
 "ok" : 1
}
```

Проверим, что команда отработала, запустим ее еще раз:

```
rs.initiate()
```

Ответ должен быть следующим:

```
{
 "info" : "try querying local.system.replset to see current
 configuration",
 "ok" : 0,
 "errmsg" : "already initialized"
}
```

Обратите внимание — приглашение показывает, что этот узел стал первичным (**Primary**):

```
myrs:PRIMARY>
```

Создадим второй узел при помощи метода **rs.add()**, который добавляет член в набор реплик. Для выполнения метода первичный узел **Primary** должен быть запущен:

```
rs.add("my_comp.localdomain:30001")
```

Ответ:

```
{ "ok" : 1 }
```

Создадим арбитра. Для этого запустим метод **rs.add()** с параметром **arbiterOnly**:

```
rs.add("my_comp.localdomain:30002", {arbiterOnly: true})
```

Ответ:

```
{ "ok" : 1 }
```


Перейдем с первичного узла на вторичный. Сперва выйдем из первичного узла:

```
myrs:PRIMARY> exit
```

Ответ:

```
bye
```

И перейдем на вторичный узел:

```
mongo --port 30001 --host my_comp.localdomain
```

Ответ показывает, что мы перешли на вторичный узел:

```
MongoDB shell version: 2.6.11
connecting to: my_comp.localdomain:30001/test
myrs:SECONDARY>
```

Проверим состояние набора реплик, используя метод **db.isMaster()**.

В полученном ответе видно, что первичным узлом является узел **"primary"**: **"my_comp.localdomain:30000"**, мы находимся на вторичном узле **"me"**: **"my_comp.localdomain:30001"** и в наборе реплик имеется арбитр **"arbiters"**: **"my_comp.localdomain:30002"**:

```
{
 "setName" : "myrs",
 "setVersion" : 3,
 "ismaster" : false,
 "secondary" : true,
 "hosts" : [
 "my_comp.localdomain:30001",
 "my_comp.localdomain:30000"
],
 "arbiters" : [
  "my_comp.localdomain:30002"
],
 "primary" : "my_comp.localdomain:30000",
 "me" : "my_comp.localdomain:30001",
 "maxBsonObjectSize" : 16777216,
 "maxMessageSizeBytes" : 48000000,
 "maxWriteBatchSize" : 1000,
 "localTime" : ISODate("2015-11-26T10:06:44.543Z"),
 "maxWireVersion" : 2,
 "minWireVersion" : 0,
 "ok" : 1
}
```

Проверим текущее состояние набора реплик, используя данные, полученные от обмена тактовыми сигналами (**heartbeats**) с другими членами набора реплик при помощи метода **rs.status()**:

Ответ:

```
{
 "set" : "myrs",
 "date" : ISODate("2015-11-26T10:07:23Z"),
 "myState" : 2,
 "syncingTo" : "my_comp.localdomain:30000",
 "members" : [
  {
   "_id" : 0,
   "name" : "my_comp.localdomain:30000",
   "health" : 1,
   "state" : 1,
   "stateStr" : "PRIMARY",
   "uptime" : 187,
   "optime" : Timestamp(1448532312, 1),
   "optimeDate" : ISODate("2015-11-26T10:05:12Z"),
   "lastHeartbeat" : ISODate("2015-11-26T10:07:22Z"),
   "lastHeartbeatRecv" : ISODate("2015-11-26T10:07:22Z"),
   "pingMs" : 0,
   "electionTime" : Timestamp(1448532051, 2),
   "electionDate" : ISODate("2015-11-26T10:00:51Z")
  },
  {
   "_id" : 1,
   "name" : "my_comp.localdomain:30001",
   "health" : 1,
   "state" : 2,
   "stateStr" : "SECONDARY",
   "uptime" : 512,
   "optime" : Timestamp(1448532312, 1),
   "optimeDate" : ISODate("2015-11-26T10:05:12Z"),
   "self" : true
  },
  {
   "_id" : 2,
   "name" : "my_comp.localdomain:30002",
   "health" : 1,
   "state" : 7,
   "stateStr" : "ARBITER",
   "uptime" : 131,
   "lastHeartbeat" : ISODate("2015-11-26T10:07:22Z"),
```



```
   "lastHeartbeatRecv" : ISODate("2015-11-26T10:07:21Z"),
   "pingMs" : 0
  }
  ],
  "ok" : 1
}
```

Вернемся на первичный узел (предварительно выйдем из вторичного узла **myrs:SECONDARY> exit**) и введем команду:

```
mongo my_comp.localdomain:30000
```

Стандартный ответ:

```
MongoDB shell version: 2.6.11
connecting to: my_comp.localdomain:30000/test
myrs:PRIMARY>
```

Создадим, как это было описано ранее, базу данных **students**:

```
use students
```

Ответ:

```
switched to db students
```

Создадим коллекцию с именем **student**:

```
db.student.insert({name: "Иванов Иван Иванович"})
```

Ответ (создан один документ):

```
WriteResult({ "nInserted" : 1 })
```

Проверим, какие базы данных существуют на данном узле:

```
show dbs
```

Ответ (имеется три базы данных: системная **admin**, журнал **local** и созданная нами **students**):

```
admin (empty)
local 1.078GB
students 0.078GB
```

Перейдем на вторичный узел (как это сделать, см. выше) и проверим, какие базы данных находятся в нем. Очевидно, что они должны полностью совпадать с базами данных первичного узла. На выполнение команды

```
myrs:SECONDARY> show dbs
```

должен прийти ответ:

```
admin (empty)
local 1.078GB
students 0.078GB
```

Переключимся на базу данных **students** (**use students**).

Выполним команду **rs.slaveOk()**, которая разрешает операцию считывания вторичным узлам.

Проверим содержимое коллекции **student**:

```
db.student.find()
```

Ответ:

```
{ "_id" : ObjectId("5656db14607ddf7a56079425"), "name" :
"Иванов Иван Иванович" }
```

Вернемся на первичный узел (через команды **exit** и **mongo my_comp.localdomain:30000**). Осуществим имитацию отказа узла, чтобы проверить, каким образом при помощи арбитра будет назначен новый первичный узел. Для этого нужно либо перейти в окно, в котором запущен узел, и нажать **Ctrl+C** (узел будет остановлен), либо использовать метод **db.shutdownServer()**. Для этого в рабочем окне перейти в режим администратора (**use admin**). После выполнения команды **db.shutdownServer()** получим ответ, который показывает, что сервер отключен:

```
2015-11-26T13:20:12.790+0300 DBClientCursor::init call() failed
server should be down...
2015-11-26T13:20:12.794+0300 trying reconnect to
my_comp.localdomain:30000 (127.0.0.1) failed
2015-11-26T13:20:12.795+0300 warning: Failed to connect to
127.0.0.1:30000, reason: errno:111 Connection refused
2015-11-26T13:20:12.795+0300 reconnect
my_comp.localdomain:30000 (127.0.0.1) failed failed couldn't
connect to server my_comp.localdomain:30000 (127.0.0.1),
connection attempt failed...
```

Выйдем (**exit**) и подключимся ко второму узлу (**mongo my_comp.localdomain:30001**). Снова проверим текущее состояние реплик (метод **rs.status()**). Из ответа сервера видно, что узел **my_comp.localdomain:30001** стал первичным (поскольку, не получая периодических сигналов от первичного узла **my_comp.localdomain:30000**, арбитр и узел **my_comp.localdomain:30001**, обмениваясь сигналами, на-

значают бывший вторичный узел на роль первичного), а узел **my_comp.localdomain:30000** становится недоступен:

```
{
 "set" : "myrs",
 "date" : ISODate("2015-11-26T10:22:30Z"),
 "myState" : 1,
 "members" : [
  {
   "_id" : 0,
   "name" : " my_comp.localdomain:30000",
   "health" : 0,
   "state" : 8,
   "stateStr" : "(not reachable/healthy)",
   "uptime" : 0,
   "optime" : Timestamp(1448532757, 1),
   "optimeDate" : ISODate("2015-11-26T10:12:37Z"),
   "lastHeartbeat" : ISODate("2015-11-26T10:22:27Z"),
   "lastHeartbeatRecv" : ISODate("2015-11-26T10:20:10Z"),
   "pingMs" : 0
  },
  {
   "_id" : 1,
   "name" : " my_comp.localdomain:30001",
   "health" : 1,
   "state" : 1,
   "stateStr" : "PRIMARY",
   "uptime" : 1419,
   "optime" : Timestamp(1448532757, 1),
   "optimeDate" : ISODate("2015-11-26T10:12:37Z"),
   "electionTime" : Timestamp(1448533219, 1),
   "electionDate" : ISODate("2015-11-26T10:20:19Z"),
   "self" : true
  },
  {
   "_id" : 2,
   "name" : " my_comp.localdomain:30002",
   "health" : 1,
   "state" : 7,
   "stateStr" : "ARBITER",
   "uptime" : 1038,
   "lastHeartbeat" : ISODate("2015-11-26T10:22:29Z"),
   "lastHeartbeatRecv" : ISODate("2015-11-26T10:22:29Z"),
   "pingMs" : 0
  }
 ],
 "ok" : 1
}
```

Теперь у нас имеется один узел и отсутствует резервирование. Это создает потенциально опасную ситуацию, поэтому запустим узел **my_comp.localdomain:30000** снова. Перейдем в окно (консоль), которое отвечало за процесс на узле **my_comp.localdomain:30000**. Запустим работу узла:

```
mongod --replSet myrs --dbpath /data/myrepl/node1 --port 30000
```

В качестве ответа получим:

```
2015-11-26T13:24:07.252+0300 [initandlisten] MongoDB starting
: pid=7014 port=30000 dbpath=/data/myrepl/node1 64-bit host=
my_comp.localdomain
2015-11-26T13:24:07.252+0300 [initandlisten] db version
v2.6.11
```

Вернемся в рабочее окно, выйдем из текущего узла (**exit**) и подключимся к только что запущенному узлу **mongo (my_comp. localdomain:30000)**.

По полученному приглашению к работе (**myrs:SECONDARY>**) очевидно, что узел стал вторичным.

Вернемся на первичный узел (команды **exit** и **mongo my_comp.localdomain:30001**), поскольку все операции по модификации данных могут производиться только на нем (попытка внести данные на вторичном узле приведет к появлению сообщения об ошибке).

Так как репликация невозможна без журнала репликаций (**oplog**), рассмотрим его подробнее. Журнал представляет из себя коллекцию ограниченного размера, которая находится в базе данных **local** на каждом узле (и является локальной для каждого узла). Когда на первичном узле происходит, например, запись, то информация о произведенной операции фиксируется в журнале. После репликации на вторичный узел информация попадает в его журнал репликаций. Для того чтобы посмотреть журнал, перейдем на базу данных **local**:

```
use local
```

Ответ узла:

```
switched to db local
```

Посмотрим, какие коллекции существуют в базе данных **local**:

```
show collections
```


Ответ узла с перечислением коллекций:

```
me
oplog.rs
replset.minvalid
slaves
startup_log
system.indexes
system.replset
myrs:PRIMARY>
```

Из перечня системных коллекций нас интересует журнал — коллекция **oplog.rs**. Более подробную информацию об остальных локальных коллекциях можно найти в соответствующей документации.

Вспомним, что к настоящему моменту была создана база данных **students**, в ней создана коллекция **student**, в которой внесена одна запись. Проверим, каким образом в журнале отражается наша операция вставки (**op: "i" — insert**). После выполнения операции:

```
db.oplog.rs.findOne({op: "i"})
```

получим ответ узла, содержащий информацию журнала по интересующей нас операции (выдаваемые поля коллекции содержат в том числе временную метку записи **ts**, тип операции **op**, пространство имен (название базы данных и коллекции) **ns**):

```
{
  "ts" : Timestamp(1448532757, 1),
  "h" : NumberLong("5819097236332024749"),
  "v" : 2,
  "op" : "i",
  "ns" : "students.student",
  "o" : {
  "_id" : ObjectId("5656db14607ddf7a56079425"),
  "name" : "Иванов Иван Иванович"
 }
}
```

Перейдем к базе данных **students**:

```
use students
```

и добавим по очереди два документа в коллекцию **student**:

```
db.student.insert({name: "Петров Петр Петрович"})
db.student.insert({name: "Сидоров Сидор Сидорович"})
```

Добавим поле с названием кафедры во все документы коллекции:

```
db.student.update({}, {$set: {cafedra: "Кафедра Алгебры"}},
false, true)
```

Поскольку были изменены три документа, то в качестве ответа получим:

```
WriteResult({ "nMatched" : 3, "nUpserted" : 0, "nModified" : 3 })
```

Проверим, что данные были внесены:

```
db.student.find()
```

Ответ:

```
{ "_id" : ObjectId("5656db14607ddf7a56079425"), "name" :
"Иванов Иван Иванович", "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("5656dfc64b076f5e4e87bb40"), "name" :
"Петров Петр Петрович", "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("5656dfe74b076f5e4e87bb41"), "name" :
"Сидоров Сидор Сидорович", "cafedra" : "Кафедра Алгебры" }
```

Вернемся на локальную базу данных (**use local**) и проверим, как отразились операции изменения в журнале. Запустим команду с аргументом (**«u» — update**):

```
myrs:PRIMARY> db.oplog.rs.find({op: "u"})
```

Результат выполнения операции — три измененных записи:

```
{ "ts" : Timestamp(1448534118, 1), "h" :
NumberLong("-6401947800642097895"), "v" : 2, "op" : "u", "ns"
: "students.student", "o2" : { "_id" :
ObjectId("5656db14607ddf7a56079425") }, "o" : { "$set" : {
"cafedra" : "Кафедра Алгебры" } } }
{ "ts" : Timestamp(1448534118, 2), "h" :
NumberLong("-2721134207572438882"), "v" : 2, "op" : "u", "ns"
: "students.student", "o2" : { "_id" :
ObjectId("5656dfc64b076f5e4e87bb40") }, "o" : { "$set" : {
"cafedra" : "Кафедра Алгебры" } } }
{ "ts" : Timestamp(1448534118, 3), "h" :
NumberLong("4096368831517265582"), "v" : 2, "op" : "u", "ns" :
"students.student", "o2" : { "_id" :
ObjectId("5656dfe74b076f5e4e87bb41") }, "o" : { "$set" : {
"cafedra" : "Кафедра Алгебры" } } }
```


Для того чтобы получить информацию о текущем состоянии журнала, используется команда:

```
db.getReplicationInfo()
```

В ответе узла можно увидеть различную служебную информацию: размер журнала, временные метки первой (**tFirst**) и последней (**tLast**) записей и пр.:

```
{
 "logSizeMB" : 990,
 "usedMB" : 0.01,
 "timeDiff" : 1864,
 "timeDiffHours" : 0.52,
 "tFirst" : "Thu Nov 26 2015 13:04:14 GMT+0300 (MSK)",
 "tLast" : "Thu Nov 26 2015 13:35:18 GMT+0300 (MSK)",
 "now" : "Thu Nov 26 2015 13:38:29 GMT+0300 (MSK)"
}
```

Заметим, что хранение данной информации необходимо как для синхронизации реплик, так и для обработки отказа. Поскольку первоначально операции производятся на первичном узле, вторичные узлы (имеющие свой журнал репликации) при синхронизации реплик запрашивают у первичного узла все записи, которые имеют более поздние метки по сравнению с метками, имеющимися в их журналах. После этого записи добавляются в журналы и применяются к базам данных на вторичных узлах. Таким образом, на каждом узле имеется последний вариант базы данных и журнала, что позволяет в случае отказа оперативно восстановить работоспособность системы, назначив в качестве первичного один из вторичных узлов.

Для поиска самой последней записи в журнале используется команда

```
db.oplog.rs.find().sort({$natural:-1}).limit(1)
```

Очевидно, что ответом будут данные об операции внесения поля с названием кафедры:

```
{ "ts" : Timestamp(1448534118, 3), "h" :
NumberLong("4096368831517265582"), "v" : 2, "op" : "u", "ns" :
"students.student", "o2" : { "_id" :
ObjectId("5656dfe74b076f5e4e87bb41") }, "o" : { "$set" : {
"cafedra" : "Кафедра Алгебры" } } }
```

Остальные вопросы, связанные с администрированием репликаций, рассмотрены не будут ввиду их сложности и объема, более подробную информацию можно найти в соответствующей литературе.

Чтобы подвести итоги, сделаем следующее замечание. Использование репликации не всегда приводит к положительному эффекту. Бывают ситуации, когда оборудование не справляется с нагрузкой, например, если приложения в основном не считывают информацию, а записывают ее. В этом случае большая работа вторичных узлов связана с обновлением реплик (обработкой записи) и невозможно осуществить балансировку нагрузки по чтению, поскольку узлы заняты другими операциями. Однако поскольку основная функция репликации — автоматическое восстановление работоспособности системы после сбоя, ее использование позволяет избежать отказовые ситуации и сделать систему более надежной.

Шардинг (сегментирование)

Шардинг является ключевой концепцией MongoDB. До сих пор каждый запущенный процесс **mongod** оперировал с полной копией базы данных, пусть даже эти копии были размещены на различных узлах (в случае репликации). Принципиальным отличием шардинга является сегментирование базы данных и размещение сегментов на различных серверах. Это особенно актуально в тех случаях, когда объем базы данных не позволяет, например, организовать хранение данных ввиду ограничений возможностей адресации или снижает эффективность работы пользователей из-за того, что сервер не справляется с нагрузкой. Важным является то, что интерфейс MongoDB позволяет приложению работать с сегментированной базой данных абсолютно точно так же, как и с базой данных, расположенной на одном сервере.

В состав сегментированного кластера входят собственно сегменты (**shards**), причем каждый из сегментов для повышения надежности состоит из набора реплик, маршрутизаторов **mongos** (процесс, перенаправляющий операции с данными к требуемому сегменту) и конфигурационных серверов, которые отвечают за конфигурацию кластера, т. е. хранят данные о расположении баз данных, коллекций, диапазонах хранящихся данных, журналы изменений. При практическом использовании СУБД MongoDB в реальной обстановке во избежание отказов и обеспечения надежности желательно иметь три конфигурационных сервера, расположенных на трех различных машинах. Однако в случае учебного примера используется один. Кроме того, для простоты сегменты будут созданы без реплик, поскольку ранее был описан процесс репликации и желающие могут произвести репликацию самостоятельно.

Создадим (как было показано ранее) директорию и поддиректории для хранения сегментов. В директории **data** создадим поддиректорию **myshard** для хранения создаваемой сегментации, а в ней поддиректории **db1** и **db2** для размещения данных сегмента 1 и сегмента 2 соответственно и **myconfig** для служебной информации о шардинге (рис. 10.3):

```
mkdir /data/myshard/db1
mkdir /data/myshard/db2
mkdir /data/myshard/myconfig
```

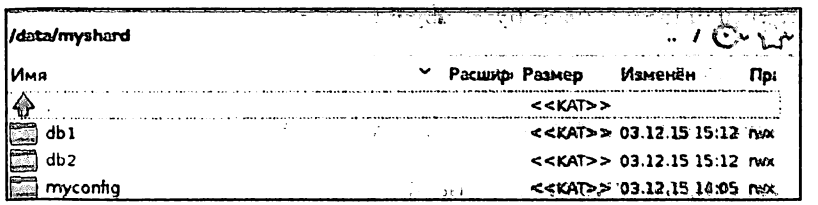


Рис. 10.3. Поддиректории для сегментирования

Далее запускаем конфигурационный сервер, используя опцию **configsvr** и задавая номер порта (по умолчанию порт 27019, но мы запустим демон на другом порту, например 27021. Можно было выбрать и любой другой из возможных портов, см. документацию). Открываем новое окно в терминале (консоли) и выполняем команду

```
mongod --configsvr --dbpath /data/myshard/myconfig --port
27021
```

Ответ сервера (ввиду большого объема приводится только начало ответа):

```
2015-12-03T13:59:34.671+0300 [initandlisten] MongoDB starting :
pid=3975 port=27 ...
```

Теперь у нас работает конфигурационный сервер, и можно запускать процесс **mongos** для настройки маршрутизации. Открываем но-

вое окно и запускаем **mongos**, при его запуске указываются имена хостов серверов конфигурации (при практическом применении трех, см. выше). Опция **configdb** в командной строке используется для того, чтобы маршрутизатор запросов **mongos** связать с конфигурационным сервером (серверами). Номера портов выбираются произвольно из разрешенного диапазона (см. документацию).

```
mongos --configdb localhost:27021 --port 40000
```

Ответ сервера (приводится часть):

```
2015-12-03T14:06:05.683+0300 warning: running with 1 config
server should be done only for testing purposes and is not
recommended for production ...
```

Создание сегментов. Для этого необходимо открыть новое окно (для каждого сегмента свое), каждый сегмент будет использовать свой собственный порт и свой путь **-dbpath**.

В первом окне запускаем:

```
mongod --port 40001 --dbpath /data/myshard/db1
```

Ответ (приводится только начало):

```
2015-12-03T14:10:06.220+0300 [initandlisten] MongoDB starting
: pid=4146 port=40001 dbpath=/data/myshard/db1 64-bit host...
```

Во втором окне запускаем:

```
mongod --port 40002 --dbpath /data/myshard/db2
```

Ответ (приводится только начало):

```
2015-12-03T14:13:15.691+0300 [initandlisten] MongoDB starting
: pid=4251 port=40002 dbpath=/data/myshard/db2 64-bit host...
```

Итак, имеется два сегмента, запущенных на локальном компьютере **localhost: 40001** и **localhost: 40002** (в результате имеем четыре открытых окна, два из которых для сегментов).

Поскольку в упомянутых выше окнах запущены необходимые для шардинга процессы, для выполнения текущих команд запускаем новое окно:

```
mongo --port 40000 --host localhost
```


Ответ (обратите внимание, что в конце приглашение идет от **mongos**):

```
MongoDB shell version: 2.6.11
connecting to: localhost:40000/test
mongos>
```

Заметим, что окна (как было сказано ранее), которые не используются в данный момент, можно перевести в фоновый режим (как это сделать — см. документацию). Но при выполнении учебного примера лучше держать все окна открытыми.

Далее запускаем метод **sh.addShard()**, который принимает на вход имя хоста (или экземпляра базы данных, или автономного набора реплик) и добавляет экземпляр базы данных или набора реплик в сегмент (в данном случае создает сегмент):

```
mongos> sh.addShard("localhost:40001")
```

Ответ:

```
{ "shardAdded" : "shard0000", "ok" : 1 }
```

Создаем второй сегмент:

```
mongos> sh.addShard("localhost:40002")
```

Ответ:

```
{ "shardAdded" : "shard0001", "ok" : 1 }
```

Затем запускаем команду **enableSharding**, которая разрешает шардинг (сегментирование) базы данных. После этого можно использовать команду **shardCollection**, чтобы начать процесс распределения данных между сегментами:

```
mongos> sh.enableSharding("students")
```

Ответ:

```
{ "ok" : 1 }
```

Теперь необходимо задать ту коллекцию, которая подлежит сегментированию, а также ключ, по которому будет производиться разделение по шардам (сегментам).

Таблица 10.1. **Аргументы метода sh.shardCollection ()**

Параметр	Тип данных	Описание
namespace	string	Пространство имен коллекции сегмента
key	document	Документ, который определяет, какой ключ использовать для разделения объектов по сегментам

Метод **sh.shardCollection (*namespace*, *key*, *unique*)** принимает следующие аргументы (табл. 10.1).

Запустим команду, в которой укажем, что в качестве коллекции используется **students.student**, а в качестве ключа для разделения шардов — идентификатор студента **student_id** (такой ключ называется сегментным ключом, сегментный ключ может являться составным, т. е. содержать несколько полей):

```
mongos> sh.shardCollection("students.student", {"student_id" :
1})
```

Ответ:

```
{ "collectionsharded" : "students.student", "ok" : 1 }
```

Переключимся на базу данных **students**:

```
mongos> use students
```

Ответ:

```
switched to db students
```

Внесем данные об Иванове Иване Ивановиче в коллекцию **student**:

```
mongos> db.student.insert({student_id: 1, name: "Иванов Иван
Иванович"})
```

Ответ:

```
WriteResult({ "nInserted" : 1 })
```

Продолжим внесение данных, по очереди введем команды:

```
mongos> db.student.insert({student_id: 2, name: "Петров Петр
Петрович"})
mongos> db.student.insert({student_id: 3, name: "Сидоров Сидор
Сидорович"})
```


Проверим текущее состояние шардинга:

```
mongos> sh.status()
```

Ответ:

```
--- Sharding Status ---
sharding version: {
 "_id" : 1,
 "version" : 4,
 "minCompatibleVersion" : 4,
 "currentVersion" : 5,
 "clusterId" : ObjectId("566021e1481ad36bcd5f7165")
}
shards:
 { "_id" : "shard0000", "host" : "localhost:40001" }
 { "_id" : "shard0001", "host" : "localhost:40002" }
databases:
{ "_id" : "admin", "partitioned" : false, "primary" : "config"
}
{ "_id" : "test", "partitioned" : false, "primary" :
"shard0000" }
{ "_id" : "students", "partitioned" : true, "primary" :
"shard0000" }
  students.student
  shard key: { "student_id" : 1 }
  chunks:
    shard0000 1
    { "student_id" : { "$minKey" : 1 } } -->> { "student_id" :
{ "$maxKey" : 1 } } on : shard0000 Timestamp(1, 0)
```

Из ответа следует, что база данных **students** может быть сегментирована (**"_id": "students", "partitioned" : true, "primary" : "shard0000"**) и доступны два сегмента (**id" : "shard0000", "host" : "localhost:40001"** и **id" : "shard0001", "host" : "localhost:40002"**). Разделение по ключу производится по идентификатору студента **student_id**.

Выйдем из процесса **mongos** и проверим, как распределились данные между сегментами:

```
exit
```

Стандартный ответ:

```
bye
```

Перейдем на первый сегмент:

```
$ mongo --host localhost --port 40001
```

Ответ:

```
MongoDB shell version: 2.6.11
connecting to: localhost:40001/test
```

Переключимся на базу данных **students**, выполнив команду **use students** (стандартный ответ: **switched to db students**). Для проверки содержимого коллекций используем метод **db.student.find()**.

Ответ:

```
{ "_id" : ObjectId("56603033d406dd166331f0d9"), "student_id" :
1, "name" : "Иванов Иван Иванович" }
{ "_id" : ObjectId("5660306dd406dd166331f0da"), "student_id" :
2, "name" : "Петров Петр Петрович" }
{ "_id" : ObjectId("56603078d406dd166331f0db"), "student_id" :
3, "name" : "Сидоров Сидор Сидорович" }
```

Очевидно, что данные внесены. Перейдем на второй сегмент, осуществив выход с первого сегмента (**exit**) и вход на второй:

```
$ mongo --host localhost --port 40002
```

Стандартный ответ:

```
MongoDB shell version: 2.6.11
connecting to: localhost:40002/test
```

Снова переключимся на базу данных **students**, выполнив команду **use students** (стандартный ответ: **switched to db students**). Для проверки содержимого коллекций используем метод **db.student.find()**. В ответ не будет выдано никакой информации, поскольку сегмент пуст.

Выйдем из данного сегмента (**exit**) и вернемся снова в процесс **mongos**:

```
$ mongo --port 40000 --host localhost
```

Переключимся на базу данных **students**, выполнив команду **use students**. Для проверки работы сегментирования необходимо внести значительное количество данных. Для этого в цикле внесем данные о кафедре (начиная со **student_id=10**, чтобы не испортить ранее внесенные данные, до **10000**). Для проверки сегментирования этого достаточно, фамилии студентов при необходимости можно внести позже:

```
mongos> var i;
mongos> for ( i = 10; i < 10000; i++ ) {
... db.student.insert({student_id: i, cafedra: "Кафедра
Алгебры" });
... }
```


Получим ответ:

```
WriteResult({ "nInserted" : 1 })
```

Вернемся на первый сегмент (команды **exit** и **$ mongo --host localhost --port 40001**) и, переключившись на базу данных **students**, проверим (команда **db.student.find()**), какие данные находятся на первом сегменте:

```
{ "_id" : ObjectId("56603033d406dd166331f0d9"), "student_id" :
1, "name" : "Иванов Иван Иванович" }
{ "_id" : ObjectId("5660306dd406dd166331f0da"), "student_id" :
2, "name" : "Петров Петр Петрович" }
{ "_id" : ObjectId("56603078d406dd166331f0db"), "student_id" :
3, "name" : "Сидоров Сидор Сидорович" }
{ "_id" : ObjectId("566031ac8e997f80578737f5"), "student_id" :
10, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f80578737f6"), "student_id" :
11, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f80578737f7"), "student_id" :
12, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f80578737f8"), "student_id" :
13, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f80578737f9"), "student_id" :
14, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f80578737fa"), "student_id" :
15, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f80578737fb"), "student_id" :
16, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f80578737fc"), "student_id" :
17, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f80578737fd"), "student_id" :
18, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f80578737fe"), "student_id" :
19, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f80578737ff"), "student_id" :
20, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f8057873800"), "student_id" :
21, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f8057873801"), "student_id" :
22, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f8057873802"), "student_id" :
23, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f8057873803"), "student_id" :
24, "cafedra" : "Кафедра Алгебры" }
```

```
{ "_id" : ObjectId("566031ac8e997f8057873804"), "student_id" :
25, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031ac8e997f8057873805"), "student_id" :
26, "cafedra" : "Кафедра Алгебры" }
Type "it" for more
```

Перейдем на второй сегмент (команды **exit** и **$ mongo --host localhost --port 40002**) и, переключившись на базу данных **students**, проверим (команда **db.student.find()**), какие данные находятся на втором сегменте:

```
{ "_id" : ObjectId("566031b38e997f8057874c5c"), "student_id" :
5233, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c5d"), "student_id" :
5234, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c5e"), "student_id" :
5235, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c5f"), "student_id" :
5236, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c60"), "student_id" :
5237, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c61"), "student_id" :
5238, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c62"), "student_id" :
5239, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c63"), "student_id" :
5240, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c64"), "student_id" :
5241, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c65"), "student_id" :
5242, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c66"), "student_id" :
5243, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c67"), "student_id" :
5244, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c68"), "student_id" :
5245, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c69"), "student_id" :
5246, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c6a"), "student_id" :
5247, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c6b"), "student_id" :
5248, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c6c"), "student_id" :
5249, "cafedra" : "Кафедра Алгебры" }
```



```
{ "_id" : ObjectId("566031b48e997f8057874c6d"), "student_id" :
5250, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c6e"), "student_id" :
5251, "cafedra" : "Кафедра Алгебры" }
{ "_id" : ObjectId("566031b48e997f8057874c6f"), "student_id" :
5252, "cafedra" : "Кафедра Алгебры" }
Type "it" for more
```

Итак, на каждом сегменте появились данные, разделение произошло по ключу **student_id**.

Для более надежной работы в условиях реальной эксплуатации масштабируемых баз данных используется репликация совместно с шардингом. Первоначально выполняется репликация базы данных, а далее производится сегментирование.

Учащимся предлагается выполнить практическую работу, основанную на изученном теоретическом материале. Выполнение практической работы предполагает изучение способов повышения надежности функционирования СУБД MongoDB при помощи репликации, а также приобретение навыков горизонтального масштабирования данных в СУБД MongoDB при помощи шардинга.

Порядок выполнения практической работы

1. Ознакомиться с описанием.
2. Создать поддиректории для хранения данных узлов репликации.
3. Создать узлы репликации (их количество и номера портов определяются полученным заданием).
4. Создать базу данных и коллекцию (в соответствии с заданием) на первичном узле. Проверить, что она реплицирована на вторичном узле.
5. Отключить первичный узел, проверить, что система работоспособна, а один из вторичных узлов принял на себя функции первичного. Внести дополнительные данные в коллекцию.
6. Включить узел, который был отключен. Проверить, что он является вторичным и данные, внесенные ранее, реплицированы.
7. Создать поддиректории для хранения данных сегментов.
8. Создать сегменты (их количество и номера портов определяются полученным заданием).
9. Создать базу данных и коллекцию (в соответствии с заданием). Проверить, что она попадает только в один сегмент.

10. Внести дополнительные данные в соответствии с заданием. Проверить, что данные распределены по различным сегментам в соответствии с ключом.

11. Сохранить результаты выполнения работы и выйти из программы.

Контрольные вопросы

1. Для чего предназначена репликация? Какие виды репликации реализованы в СУБД MongoDB?
2. Из каких шагов состоит процесс создания репликации типа **Replica Sets**? Из какого минимального количества членов она состоит?
3. Сколько процессов **mongod** необходимо открыть для создания репликации?
4. Для чего служит первичный узел репликации? Каким образом информация реплицируется на вторичный узел?
5. Что произойдет, если первичный узел будет остановлен? Запущен снова?
6. Для чего предназначен шардинг (сегментирование)?
7. Из каких шагов состоит процесс создания сегментов? Сколько процессов **mongod** и **mongos** необходимо открыть для создания сегментации?
8. Запуск какой команды разрешает шардинг (сегментирование) базы данных?
9. Каким образом осуществляется сегментирование коллекции базы данных? Перечислите основные шаги.
10. Каким образом можно повысить отказоустойчивость сегментов базы?

Часть III
ПРИМЕРЫ ПРАКТИЧЕСКОГО ИСПОЛЬЗОВАНИЯ SQL И NOSQL БАЗ ДАННЫХ

Введение

Представленные в главах 11, 12 и 13 примеры использования SQL и NoSQL баз данных можно использовать при построении информационных систем, сайтов, находящих широкое практическое применение в учебных заведениях.

В главе 11 представлена технология работы и приведены практические примеры работы в универсальном средстве DBeaver, предназначенном для одновременной работы с SQL и NoSQL базами данных.

В примере, использующем SQL базу данных MariaDB (глава 12), рассмотрено проектирование информационной системы, с помощью которой осуществляется запись на лабораторные и практические занятия в химическую, физическую лаборатории, а также в компьютерный класс.

В примере, использующем NoSQL базу данных MongoDB (глава 13), разрабатывается информационная система, с помощью которой хранятся данные о преподаваемых предметах, темах, которые необходимо изучить, а также контрольные вопросы к ним.

Лабораторная работа 11
DBEAVER — УНИВЕРСАЛЬНОЕ СРЕДСТВО ДЛЯ РАБОТЫ С SQL И NOSQL БАЗАМИ ДАННЫХ

Цель: изучение способов работы DBeaver с СУБД MariaDB и MongoDB, изучение этапов работы с ними — от создания соединения до заполнения таблиц.

В настоящее время при создании приложений, особенно для обработки разнородной информации, приходится использовать различные базы данных, как SQL-типа, так и NoSQL. До недавнего времени для работы с любой СУБД необходимо было установить свою собственную графическую оболочку (например, MySQL Workbench для СУБД MySQL, Robomongo для СУБД MongoDB) или использовать командную строку.

Мультплатформенный универсальный (работает с большим числом СУБД, как SQL, так и NoSQL) менеджер баз данных DBeaver [13] является именно тем инструментом, который в значительной степени облегчает проектирование, реализацию, модификацию и работу с различными СУБД. DBeaver является СПО и распространяется под лицензией GNU GPL2, написан на языке Java, в его основе лежит платформа Eclipse, совместим практически со всеми операционными системами семейства: Windows, Linux, Mac OS, Solaris (x86), поскольку имеются 32- и 64-битные версии. Имеется два варианта DBeaver: Community editions и Enterprise editions. Enterprise Edition включает поддержку СУБД NoSQL.

Для установки DBeaver достаточно выбрать соответствующий вариант (Community editions или Enterprise editions), скачать бесплатную версию DBeaver через Интернет и следовать инструкциям с сайта.

Говоря о функциональных возможностях, заметим, что при помощи DBeaver могут быть выполнены все основные действия с базой данных, для вызова операций имеется хорошо сгруппированное меню, кнопки на панели инструментов, а наиболее важные функции можно вызвать при помощи горячих клавиш. Основные компоненты:

- диспетчер соединений позволяет установить соединение с базой данных, выбрав требуемый драйвер. Если соединений много, то их можно организовать в папки, что особенно удобно, когда в одной папке сгруппированы соединения, используемые одним приложением;
- браузер метаданных, который показывает соединения и их содержимое, с его помощью можно просмотреть существующие таблицы, представления, столбцы, индексы, процедуры, триггеры, настройки безопасности (пользователи, роли и т. д.);
- редактор SQL, в котором можно создавать (или импортировать готовые) скрипты, редактировать SQL-запросы, создавать шаблоны и т. п.;
- редактор представлений, позволяющий выбрать способ представления данных (JSON, plain-text, XML), прокрутку большого текста, настроить работу с типами данных BLOB/CLOB (режим просмотра и редактирования), фильтрацию данных и т. п.;
- поиск данных/метаданных осуществляет либо полнотекстовый поиск, либо по заранее заданным условиям;
- сравнение структуры баз данных (сравнению подлежат только однотипные объекты: таблицы, схемы, базы данных целиком);
- импорт/экспорт баз данных, поддерживаемые форматы файлов: CSV, HTML, XML;
- ER-диаграммы (создаются автоматически), диаграмма может быть экспортирована в файлы следующих форматов: GIF, PNG, BMP, GraphML. Заметим, что диаграмма может быть создана также и для NoSQL баз данных, коллекция будет представлена как таблица, а документ — как строка таблицы;
- менеджер запросов, который позволяет сохранять все ранее выполненные запросы и статистику их выполнения (время выполнения, количество принесенных/обновленных строк, ошибки и т. д.).

Также имеется возможность администрирования пользователей (создать, удалить, изменить данные пользователя, назначить ему привилегии), посмотреть системную информацию и произвести настройки.

Ниже будут рассмотрены принципы работы DBeaver с СУБД MariaDB и MongoDB, показаны этапы работы с ними от создания соединения до заполнения таблиц.

Установка DBeaver

Для установки DBeaver необходимо зайти на официальный сайт **http://dbeaver.jkiss.org**, скачать требуемую версию (рис. 11.1, 11.2), распаковать ее в выбранной директории и запустить на выполнение (рис. 11.3). Используемая операционная система — Linux. Поскольку предусмотрено создание как SQL, так и NoSQL баз данных, то следует выбрать Enterprise Edition и версию 32 или 64 bit в зависимости от имеющегося в распоряжении компьютера (см. рис. 11.2).

Войдем в Krusader, распакуем архив и запустим на выполнение соответствующий файл (рис. 11.3).

Появится заставка (рис. 11.4).

Затем откроется окно разработчика (рис. 11.5). Поскольку на данном этапе у нас отсутствуют какие-либо соединения, то его предлагается создать.

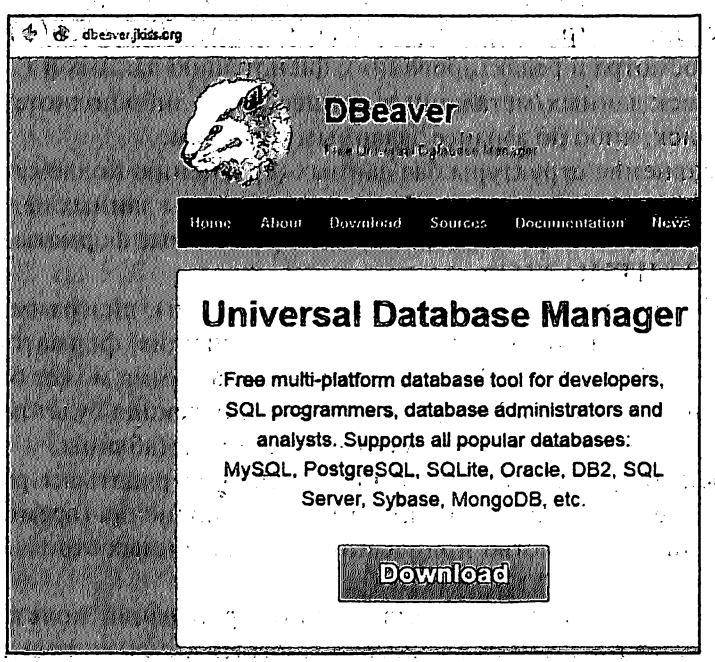


Рис. 11.1. Сайт DBeaver

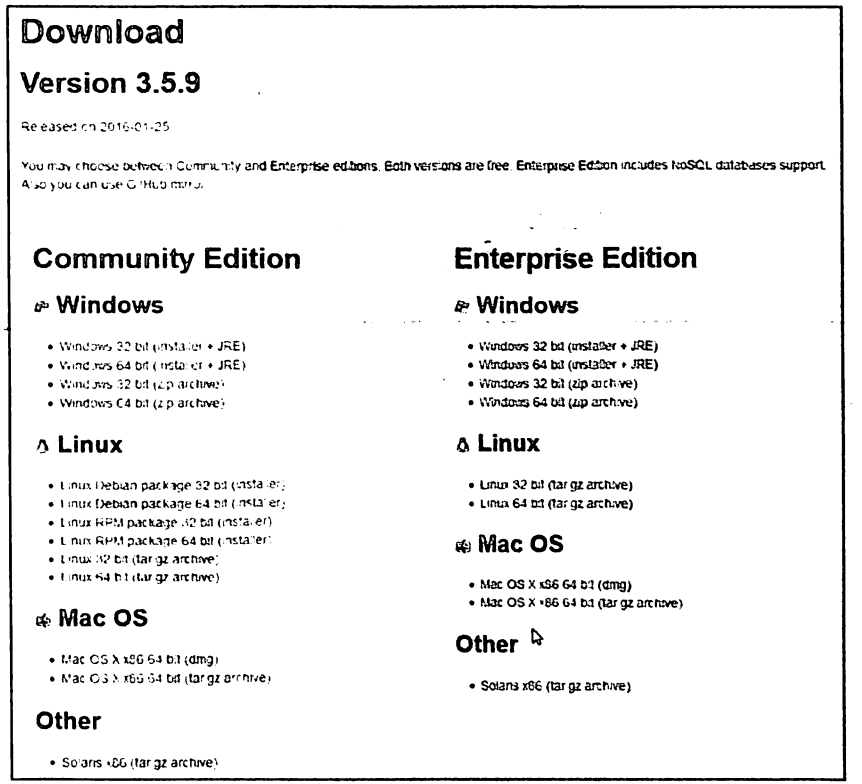


Рис. 11.2. Сайт DBeaver, выбор версии для установки

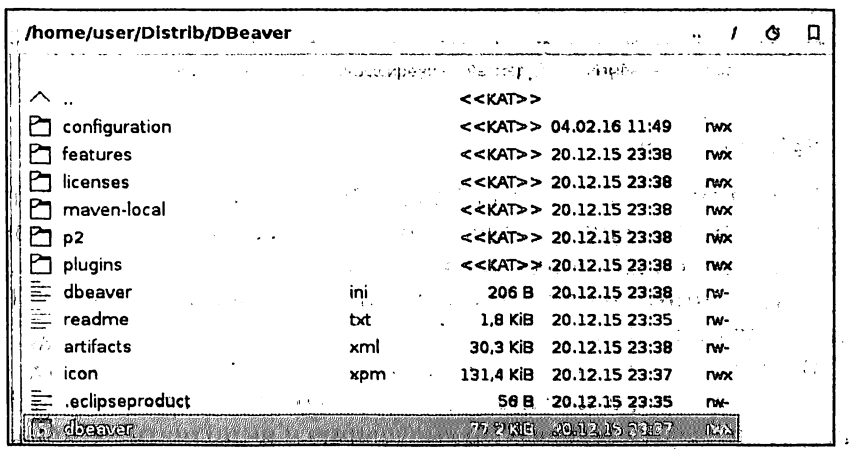


Рис. 11.3. Запуск DBeaver

Рис. 11.4. Заставка DBeaver

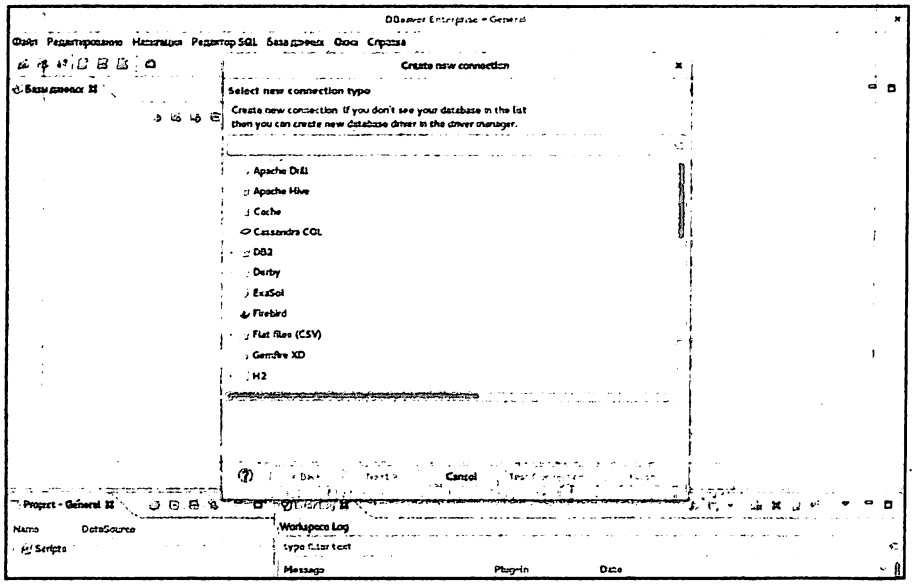


Рис. 11.5. Окно разработчика DBeaver

Работа с MariaDB

Создадим соединение для СУБД MariaDB. Выберем соответствующий элемент списка (рис. 11.6), если необходимо установить драйвер, нажмем на кнопку **Download** (рис. 11.7).

После этого проверим соединение, нажав на кнопку Test Connection (рис. 11.8).

В случае успешного соединения получим отклик (рис. 11.9).

Далее будет предложено настроить прокси-сервер (рис. 11.10). Настройки следует оставить по умолчанию и выполнять их только в случае, если пользователь точно знает результат их изменения.

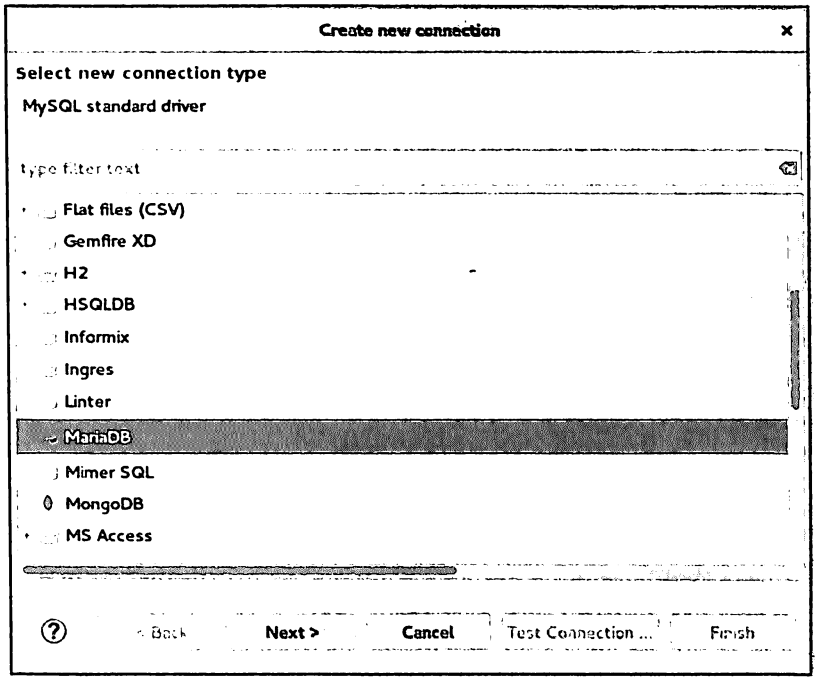


Рис. 11.6. Создание соединения с MariaDB

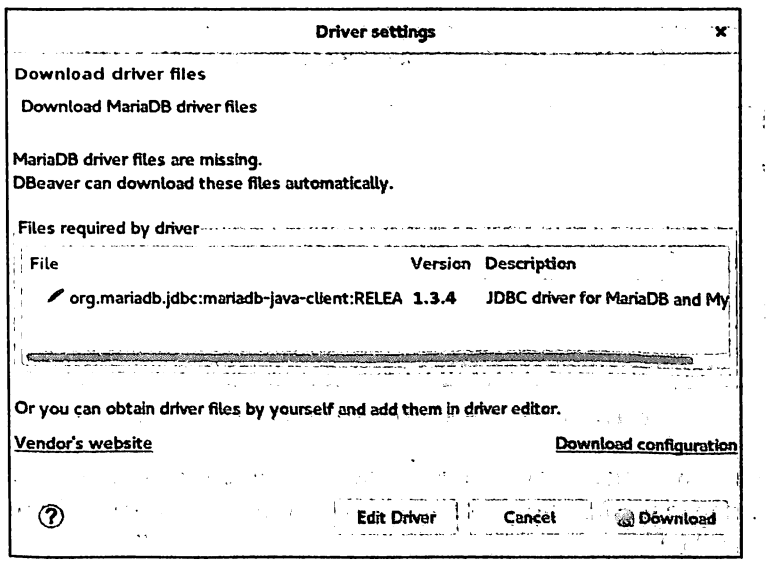


Рис. 11.7. Установка драйвера для работы с MariaDB

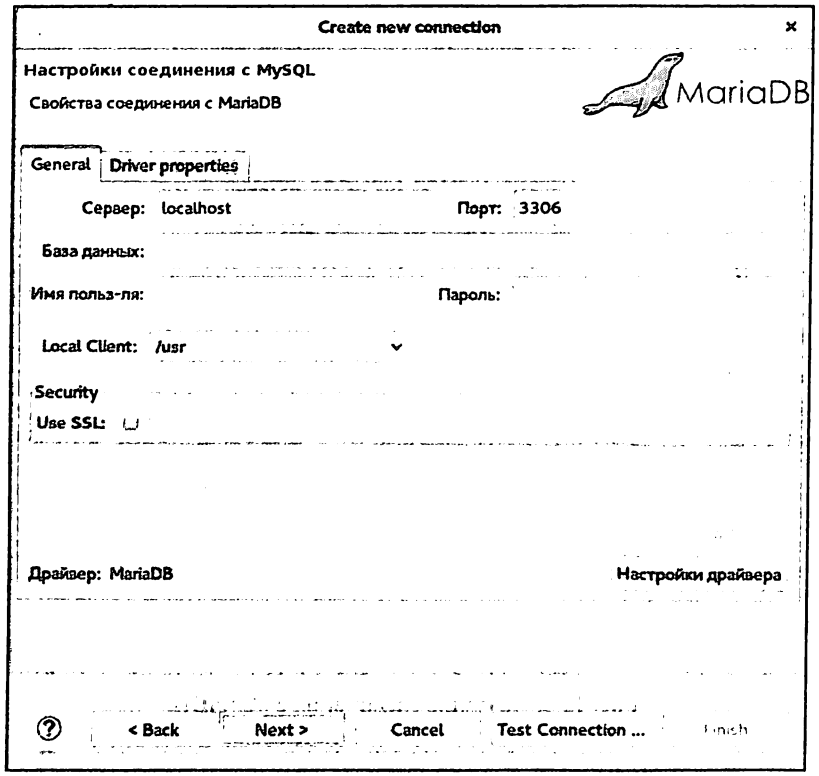


Рис. 11.8. Проверка установки соединения с MariaDB

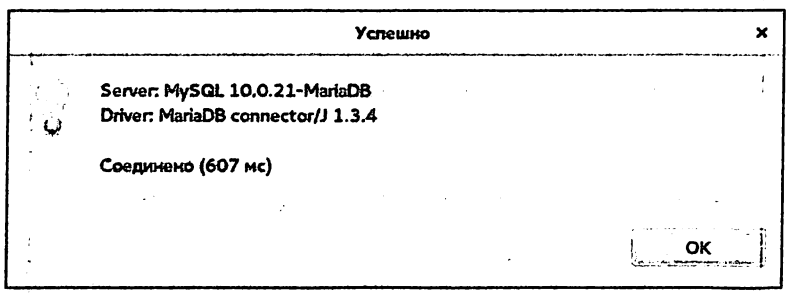


Рис. 11.9. Соединение с MariaDB установлено успешно

В завершение процесса создания соединения можно задать его имя и произвести дополнительные настройки, например безопасности (рис. 11.11).

Новое соединение создано (рис. 11.12).

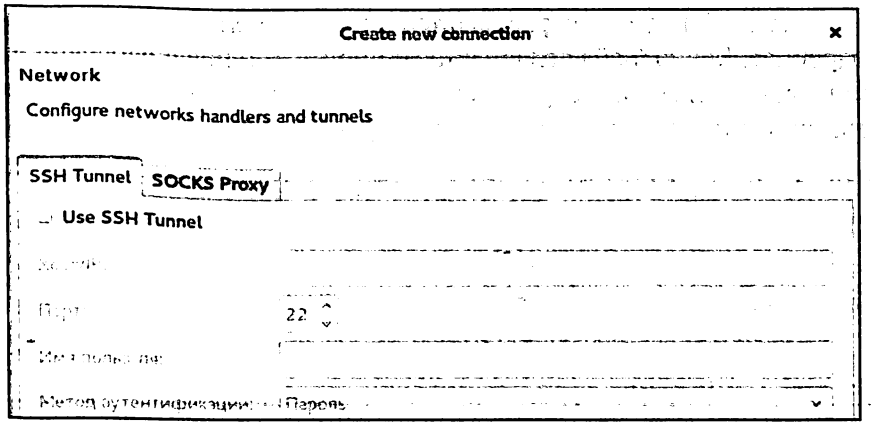


Рис. 11.10. Настройки прокси-сервера

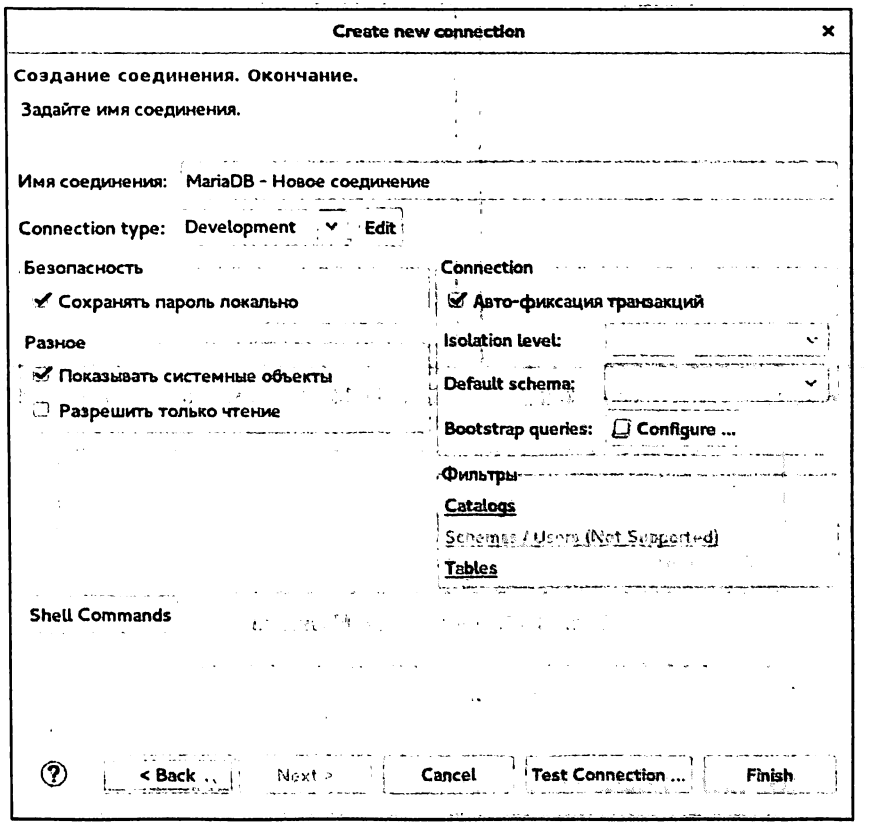


Рис. 11.11. Завершение создания соединения

В верхней части окна расположены пункты меню. Рассмотрим их подробнее. Меню **Файл** является традиционным и предназначено для стандартных операций с файлами: создание, сохранение, печать, переименование, импорт, экспорт и т. п. (рис. 11.13).

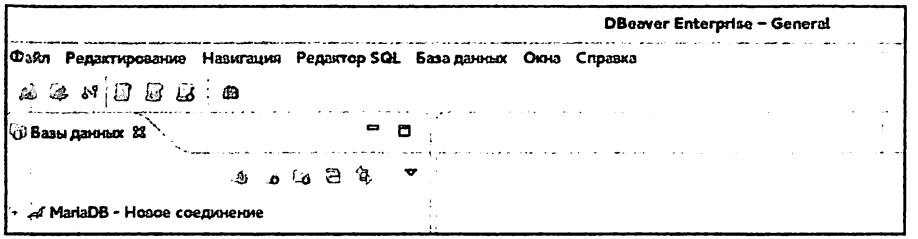


Рис. 11.12. Соединение создано

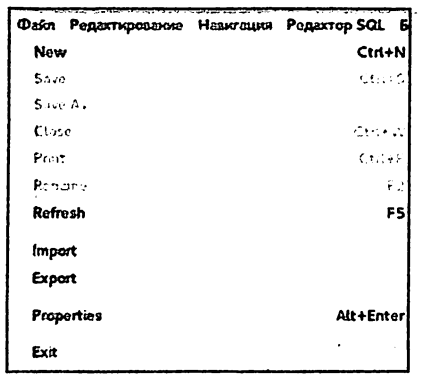


Рис. 11.13. Пункт меню **Файл**

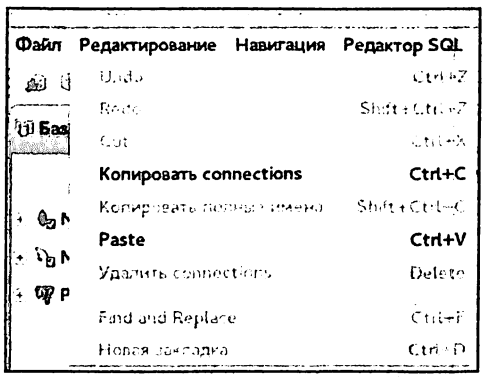


Рис. 11.14. Пункт меню **Редактирование**

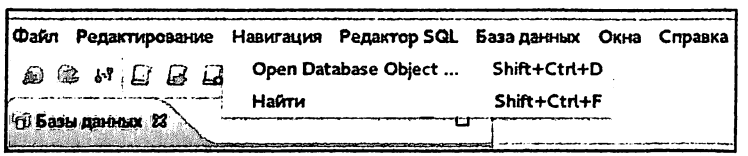


Рис. 11.15. Пункт меню **Навигация**

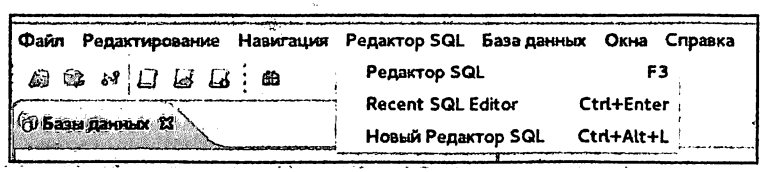


Рис. 11.16. Пункт меню **Редактор SQL**

Пункт меню **Редактирование** также является традиционным и предназначен для отмены или возврата действия, вырезания, копирования, вставки и пр. (рис. 11.14).

Для поиска базы данных используется пункт меню **Навигация** (рис. 11.15).

Для вызова и работы с редактором SQL используется соответствующий пункт меню (рис. 11.16).

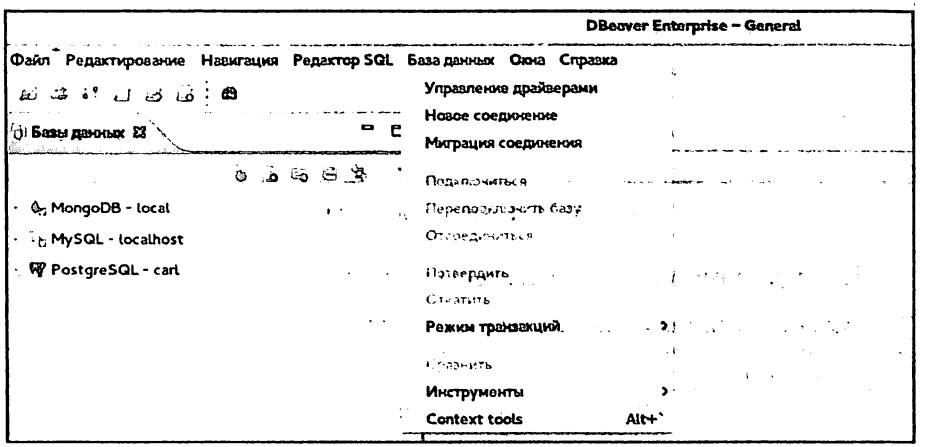


Рис. 11.17. Пункт меню **База данных**

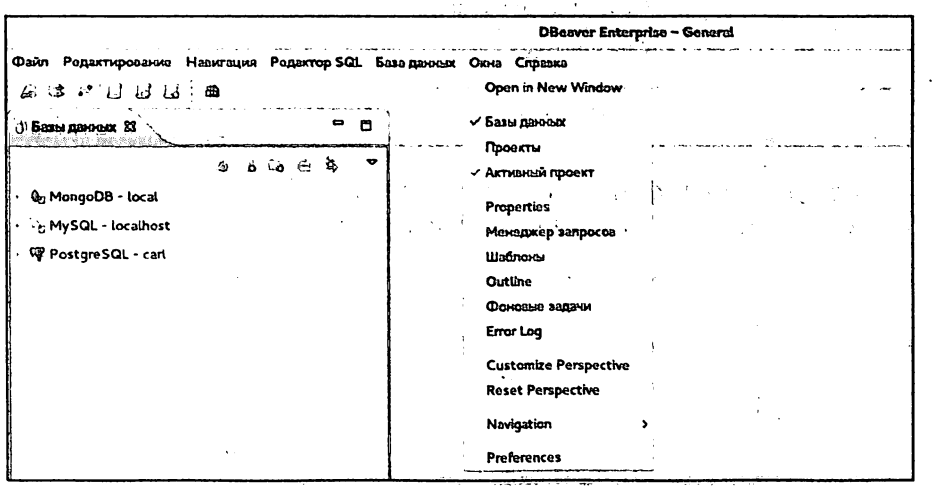


Рис. 11.18. Пункт меню **Окна**

Следующий пункт меню **База данных** предназначен для работы с базами данных, управления драйверами, создания нового соединения и т. п. (рис. 11.17).

Выбрать окна экрана можно при помощи пункта меню **Окна** (рис. 11.18).

В пункте меню **Справка** можно получить справочную информацию, а также проверить наличие обновлений (рис. 11.19).

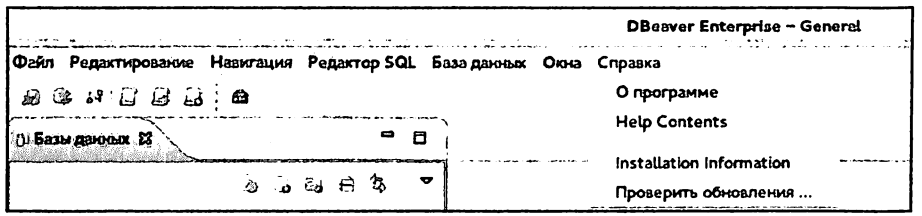


Рис. 11.19. Пункт меню **Справка**

Под пунктами меню расположены элементы управления (табл. 11.1).

Таблица 11.1. **Элементы управления панели Dbeaver**

Элемент	Значение
	Управление драйверами
	Новое соединение
	Создание новой папки
	Свернуть все
	Связь с редактором

Все указанные функции могут быть вызваны также из меню, которое откроется при нажатии на значок ▾ (рис. 11.20).

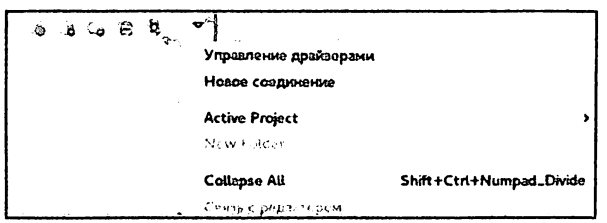


Рис. 11.20. Раскрывающееся меню

Итак, создано соединение с MariaDB. Слева в окне, которое называется **Базы данных**, показаны все действия, которые можно совершать с базой данных (рис. 11.21).

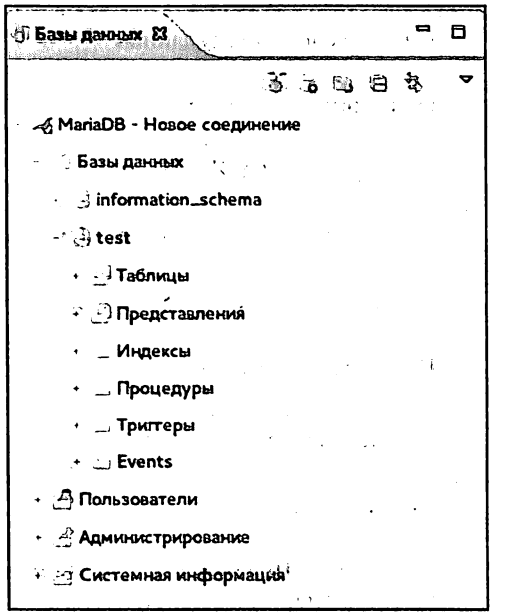


Рис. 11.21. Окно **Базы данных**

В верхней части экрана имеется панель инструментов (рис. 11.22), значения элементов которой представлены в табл. 11.2.

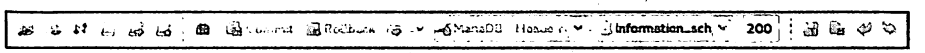


Рис. 11.22. Панель инструментов

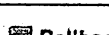
Создадим новую базу данных. Вызовем контекстное меню (правой кнопкой мыши) для объекта **Базы данных** и выберем пункт меню **Создать База данных** (рис. 11.23).

Назовем базу данных **test**, выберем из раскрывающегося списка кодировку (рис. 11.24).

В созданной базе данных создадим таблицу, вызвав соответствующий пункт контекстного меню (рис. 11.25).

Зададим имя таблицы table1 и выберем кодировку из раскрывающегося списка (рис. 11.26).

Таблица 11.2. **Элементы управления панели соединения**

Элемент	Значение
	Подключиться к базе данных
	Переподключить базу данных
	Отсоединиться от базы данных
	Вызов редактора SQL (F3)
	Вызов предыдущего редактора SQL (Ctrl+Enter)
	Вызов нового редактора SQL (Ctrl+Alt+L)
	Поиск (Shift+ Ctrl+F)
 Commit	Подтверждение транзакции
 Rollback	Откатить изменения
	Автокоммит. При нажатии этой кнопки все транзакции подтверждаются автоматически
	Просмотр/подтверждение изменений (Ctrl+S)
	Отменить все изменения
	Отменить последнюю операцию (Ctrl+Z)
	Вернуть последнюю отмененную операцию (Shift+Ctrl+Z)

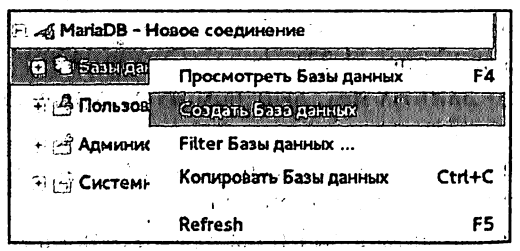


Рис. 11.23. Контекстное меню **Базы данных**

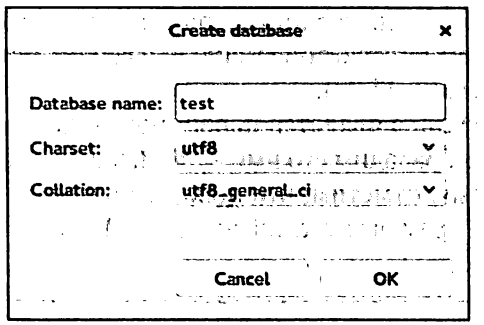


Рис. 11.24. Выбор названия и кодировки базы данных

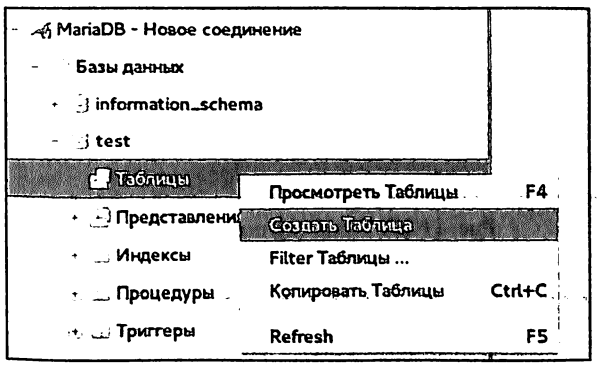


Рис. 11.25. Пункт меню **«Создать Таблица»**

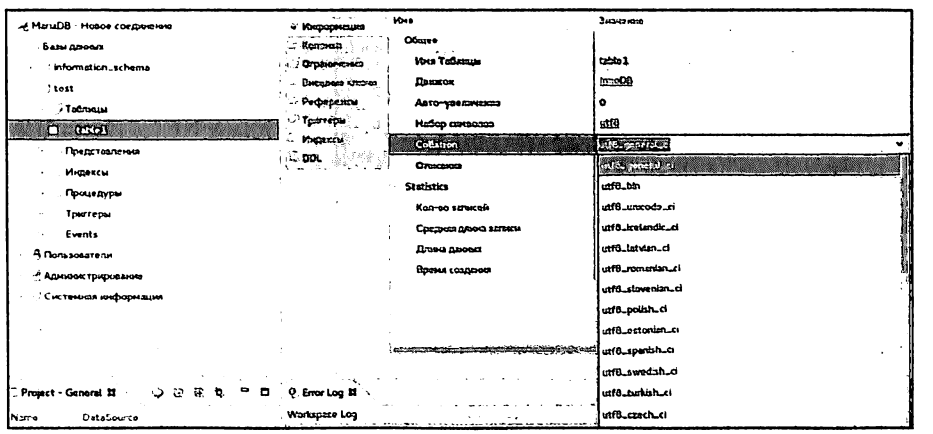


Рис. 11.26. Создание таблицы, выбор названия и кодировки

Далее создадим поля таблицы. Для этого перейдем в пункт меню **Колонки** и вызовем контекстное меню. Создадим поле, выбрав пункт **Создать Колонка** (рис. 11.27).

Пусть первое поле является первичным ключом — уникальным идентификатором. Зададим его имя — **id**. Выберем в раскрывающемся списке тип данных, отметим, что поле не является неопределенным (Не NULL) и выберем автоувеличение (рис. 11.28).

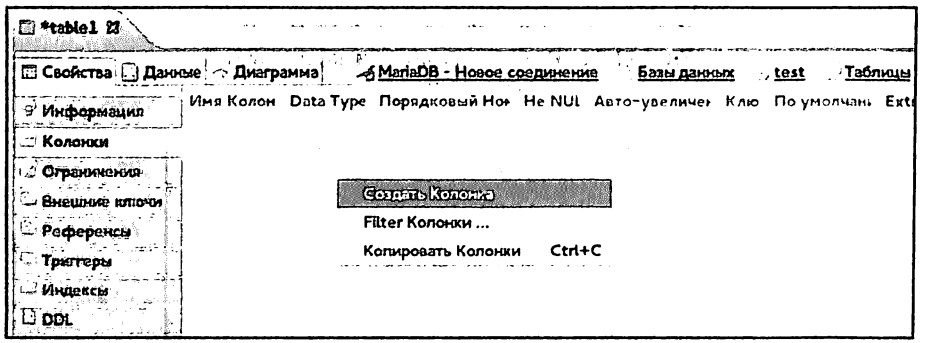


Рис. 11.27. Создание полей таблицы

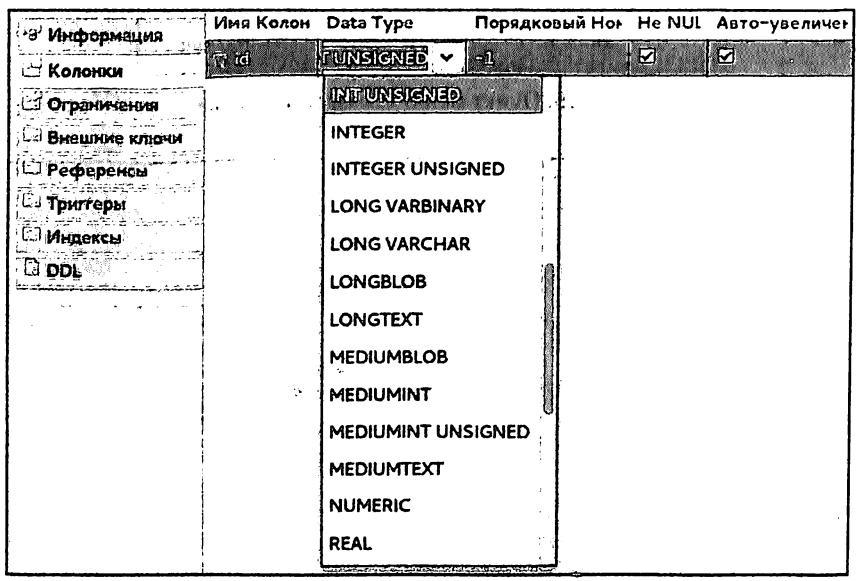


Рис. 11.28. Задание свойств поля таблицы

Далее необходимо это поле сделать первичным ключом. Войдем в пункт **Ограничения**, выберем тип **PRIMARY KEY** и поставим галочку (рис. 11.29).

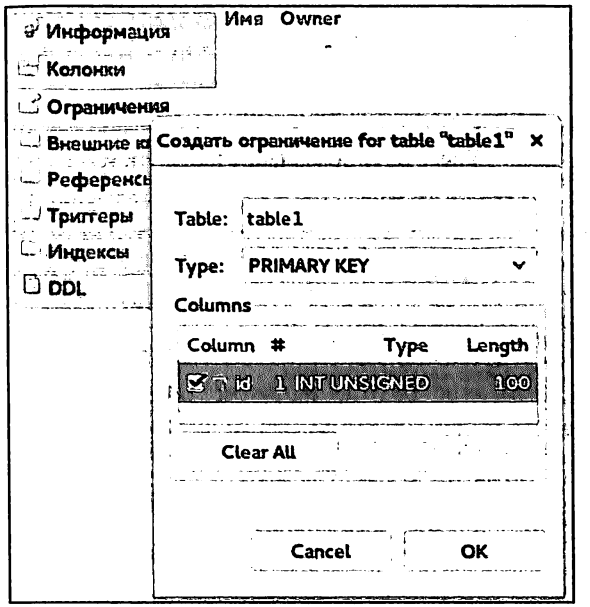


Рис. 11.29. Создание первичного ключа

Создадим второе поле с именем **Column2** и типом данных **varchar(100)** (рис. 11.30), которое не будет являться ключевым.

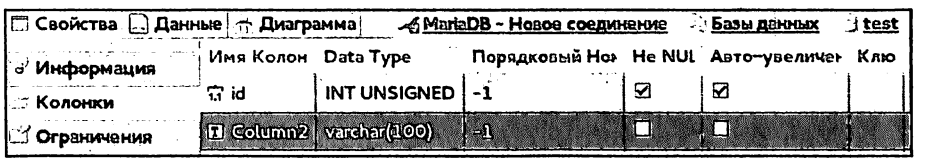


Рис. 11.30. Создание второго поля в таблице

Для сохранения результата в базе данных необходимо нажать на кнопку просмотра и сохранения изменений (см. табл. 11.2) **View/Persist Changes** (рис. 11.31).

На экране появится SQL-код, который будет сохранен в директории, в которой в соответствии с настройками базы сохраняются файлы (рис. 11.32).

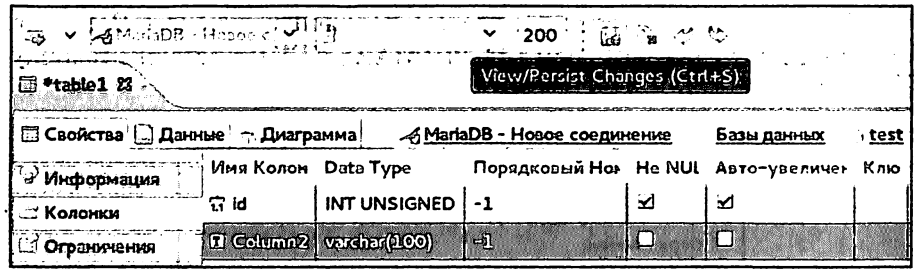


Рис. 11.31. Сохранение изменений

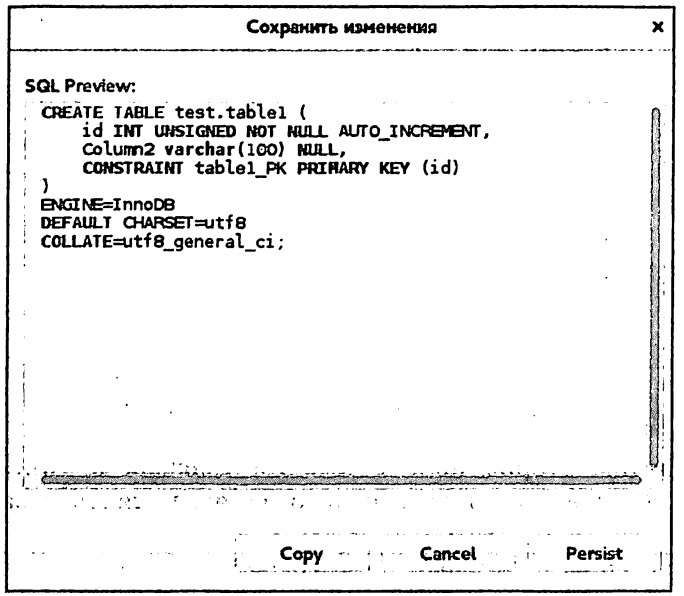


Рис. 11.32. SQL-код базы данных

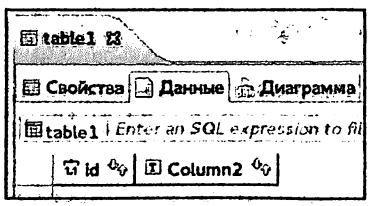


Рис. 11.33. Таблица **table1** создана

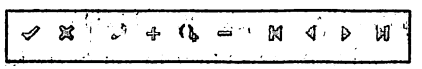


Рис. 11.34. Панель управления вкладки **Данные**

Нажимаем на кнопку **Просмотр/Сохранение изменений (View/Persist Changes)**, и таблица **table1** создана.

Перейдем на вкладку **Данные** (рис. 11.33).

Для занесения информации в таблицу будем использовать панель управления внизу вкладки **Данные** (рис. 11.34).

Значения элементов управления представлены в табл. 11.3.

Таблица 11.3. **Элементы управления вкладки Данные**

Элемент	Значение
	Применить изменение данных (Ctrl+Alt+S)
	Отменить изменение данных (Ctrl+Alt+L)
	Редактировать ячейку (Shift+Enter)
	Добавить новую запись (Alt+Insert)
	Копировать текущую запись (Ctrl +Alt+Insert)
	Удалить текущую запись (Alt+Delete)
	Переход к первой записи (Shift+Ctrl+Left)
	Переход к предыдущей записи (Ctrl+Left)
	Переход к следующей записи (Ctrl+Right)
	Переход к последней записи (Shift+Ctrl+Right)

Нажимаем на элемент управления **Добавить новую запись** (см. табл. 11.3) и в появившихся ячейках (рис. 11.35) вводим данные. Заметим, что поскольку поле **id** (первичный ключ) у нас является автоинкреметом, его заполнение происходит автоматически, и во время внесения данных оно остается пустым (рис. 11.36).

После заполнения ячеек нажимаем на кнопку **Применить изменение данных**, и все изменения сохраняются (рис. 11.37).

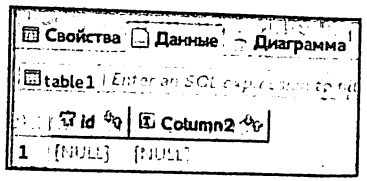


Рис. 11.35. Ячейки для внесения данных

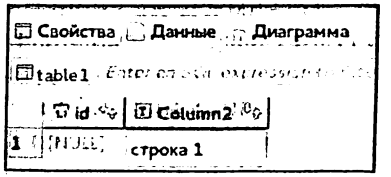


Рис. 11.36. Заполнение ячеек

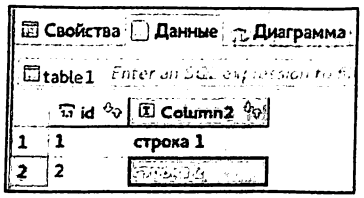


Рис. 11.37. Данные внесены

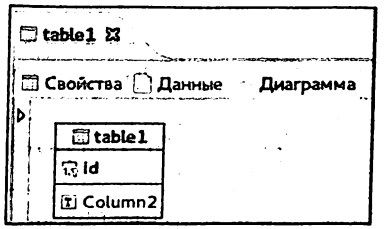


Рис. 11.38. Вкладка **Диаграммы**

Если перейти на вкладку **Диаграмма**, то увидим, что на ней появилась созданная таблица **table1** в нотации IDEF1X (рис. 11.38).

Также в MariaDB доступны возможности для работы с пользователями, администрирования и системная информация. Для этого следует выбрать соответствующий пункт основного меню и далее вызвать контекстное меню. Так, для создания нового пользователя выбирается пункт меню **Пользователи** и в контекстном меню пункт **Создать Пользователь** (рис. 11.39).

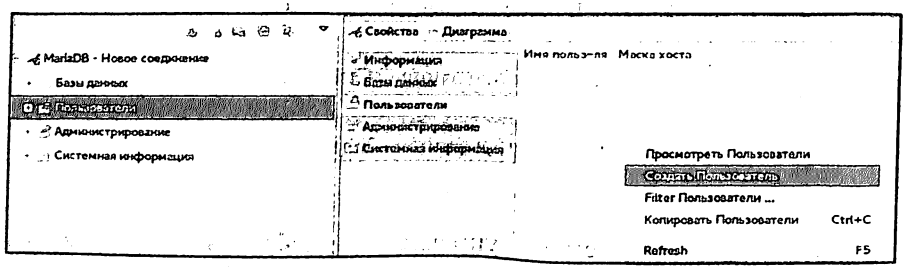


Рис. 11.39. Создание нового пользователя

Выше были подробно рассмотрены принципы администрирования и работы с пользователями, поэтому дальнейшее освоение возможностей DBeaver остается в качестве самостоятельной работы и в данном учебном пособии рассмотрено не будет.

Работа с MongoDB

До начала работы следует запустить сервер MongoDB.

Как и в случае использования MariaDB, для начала работы необходимо создать соединение. Этот этап состоит из таких же шагов, как и в предыдущем случае. Переходим на вкладку **Базы данных** и из контекстного меню выбираем пункт **Создать Соединение** (рис. 11.40). Далее выбирается соединение с MongoDB (рис. 11.41), если необходимо, то настраиваются параметры, но в общем случае следует оставить настройки по умолчанию (рис. 11.42) и производится тестовая проверка соединения (рис. 11.43).

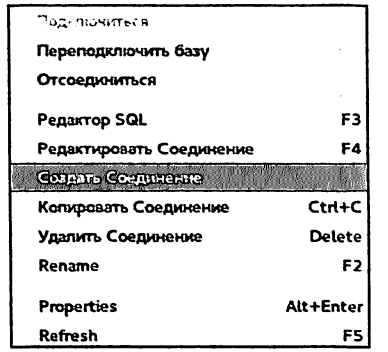


Рис. 11.40. Создание нового соединения

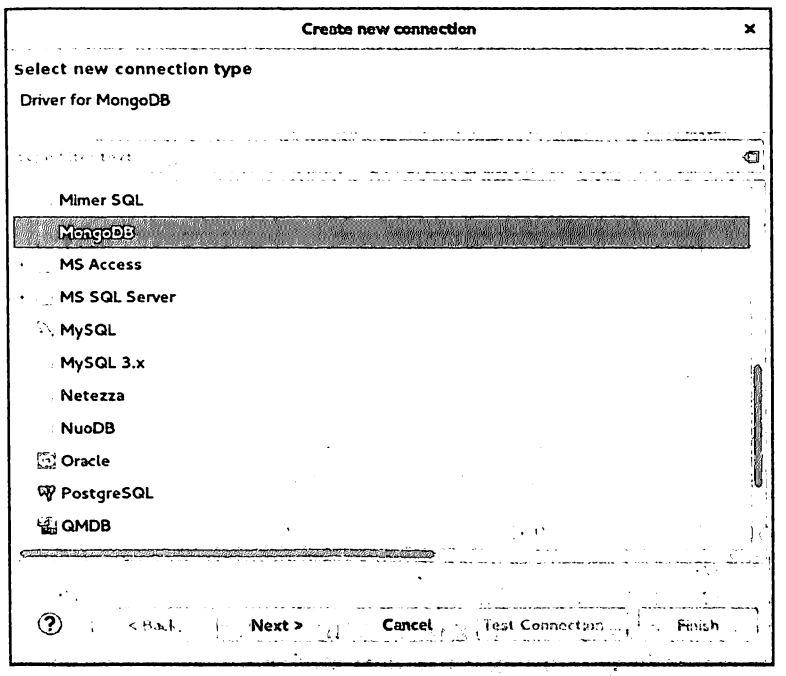


Рис. 11.41. Выбор базы для соединения

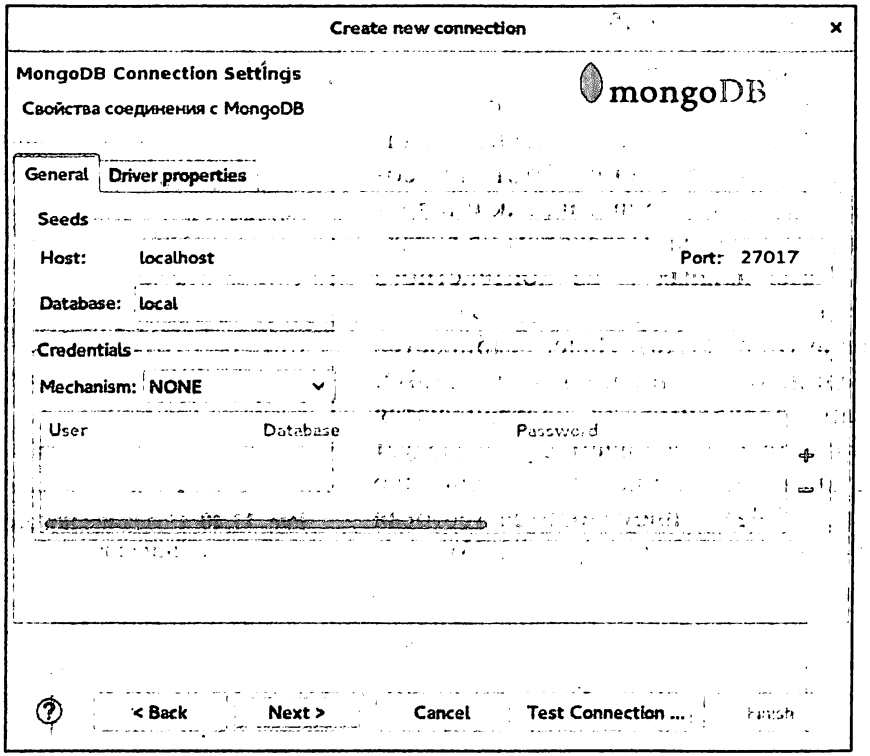


Рис. 11.42. Настройка параметров соединения

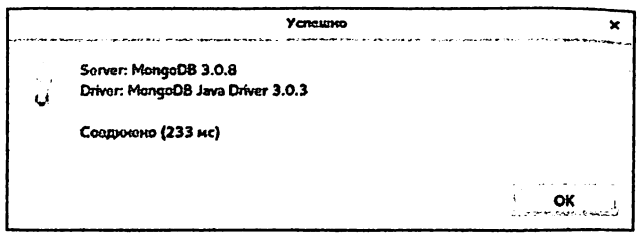


Рис. 11.43. Тестовое соединение

После этого во вкладке **Базы данных** появится новое соединение (рис. 11.44).

Перейдем к созданию базы данных. Процесс создания полностью аналогичен процессу создания базы данных MariaDB.

Создадим базу данных с именем **students** и в ней коллекцию с именем **student**. Для этого нажмем на знак «+», чтобы раскрыть спи-

Рис. 11.44. Новое соединение во вкладке **Базы данных**

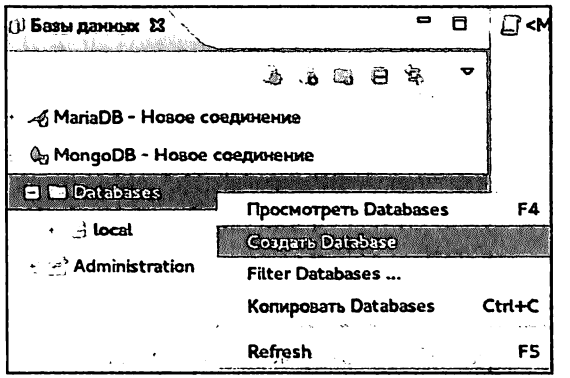


Рис. 11.45. Создание новой базы данных

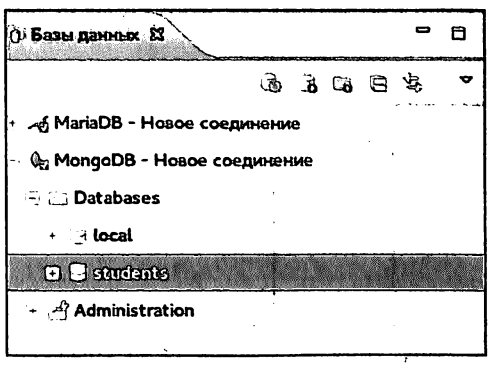


Рис. 11.46. Новая база данных создана

сок опций соединения, и вызовем пункт **Создать Database** в контекстном меню пункта **Databases** (рис. 11.45).

В раскрывающемся окне внесем имя базы данных, которое сразу же появится в раскрывающемся списке баз данных (рис. 11.46).

Далее создадим коллекцию, выбрав пункт **Создать Collection** контекстного меню (рис. 11.47) и в открывшемся окне внесем название коллекции (рис. 11.48).

Название новой коллекции появится в раскрывающемся списке (рис. 11.49.

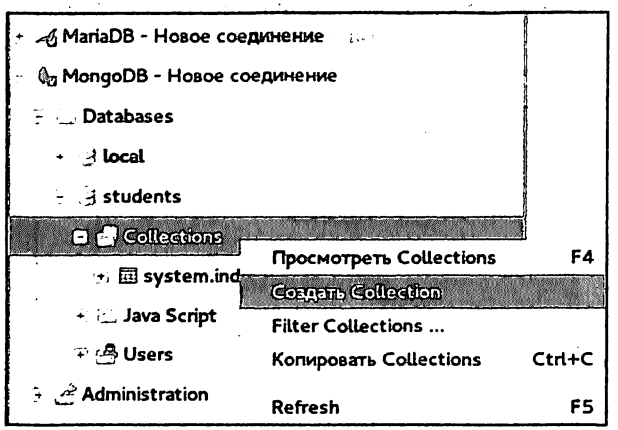


Рис. 11.47. Создание новой коллекции

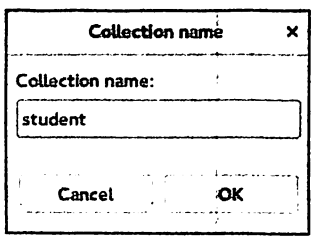


Рис. 11.48. Внесение названия новой коллекции

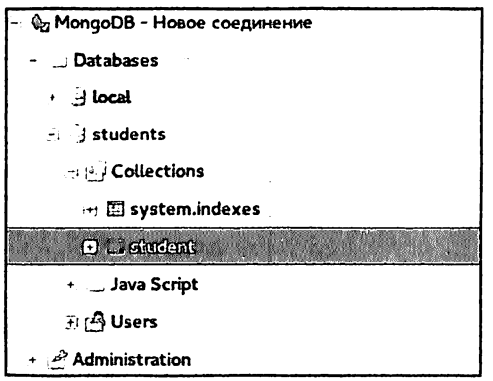


Рис. 11.49. Новая коллекция создана

Далее базу данных можно использовать для занесения данных. Сделаем это через скрипт (см. выше, каким образом работает операция **insert**). Для этого необходимо сделать базу данных активной, т. е. переключиться на нее (рис. 11.50).

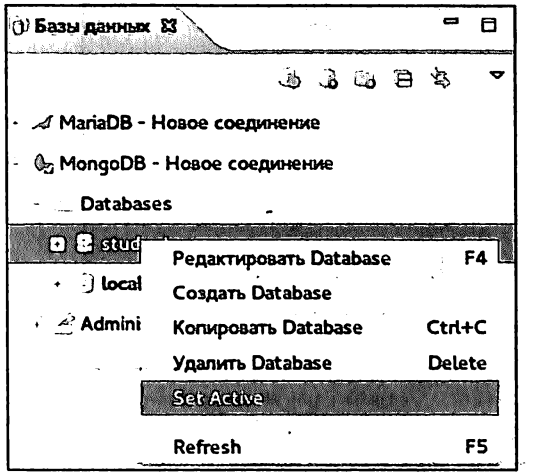


Рис.11. 50. Переключение на выбранную базу данных

Перейдем на вкладку **Project — General** внизу экрана и вызовем контекстное меню скриптов. Выберем пункт Create SQL Script (рис. 11.51).

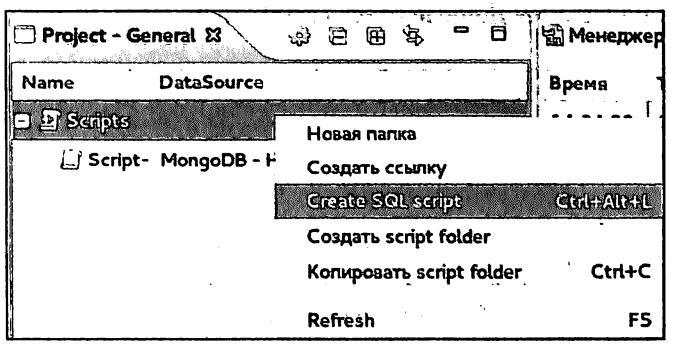


Рис. 11.51. Создание нового скрипта

В открывшемся окне выберем соединение (рис. 11.52).

В основном окне введем код скрипта (см. выше, как производится запись в базу данных MongoDB) (рис. 11.53).

Для запуска скрипта на исполнение выберем пиктограмму на панели управления выполнения скриптов (рис. 11.54) или нажмем Ctrl+Enter.

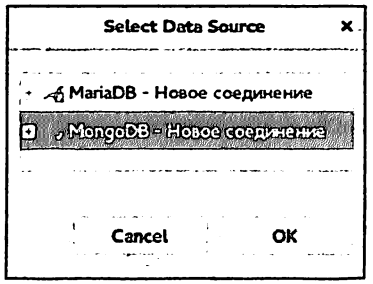


Рис. 11.52. Выбор соединения

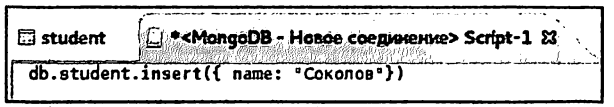


Рис. 11.53. Внесение данных в коллекцию **student**

Рис. 11.54. Панель управления выполнения скриптов

После выполнения скрипта проверим, что информация внесена. Перейдем на вкладку **Данные** (рис. 11.55).

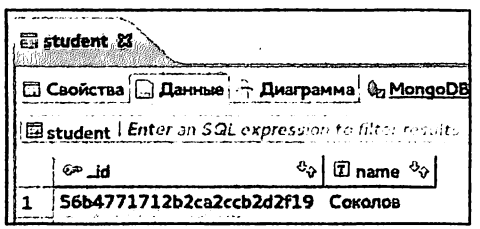


Рис. 11.55. Данные внесены

Аналогичным образом внесем данные о Петрове. Обратим внимание, что в менеджере запросов в нижней части экрана отображается информация о наших действиях (рис. 11.56).

Посмотрим результат выполнения (рис. 11.57).

Обратите внимание, что коллекция представляется в привычном для пользователей реляционных баз данных виде.

Если перейти на вкладку **Диаграмма**, то на ней коллекции также будут представлены в виде ER-диаграмм, т. е. и в этом случае отсутствуют различия в представлении диаграмм SQL и NoSQL баз данных (рис. 11.58).

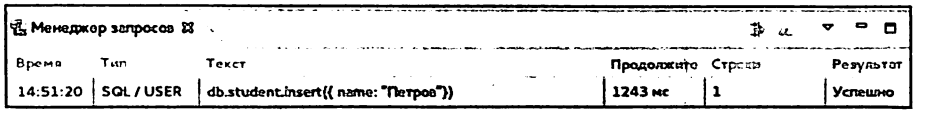


Рис. 11.56. Менеджер запросов

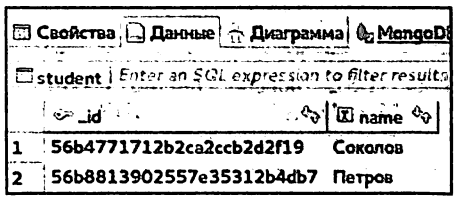


Рис. 11.57. Данные внесены

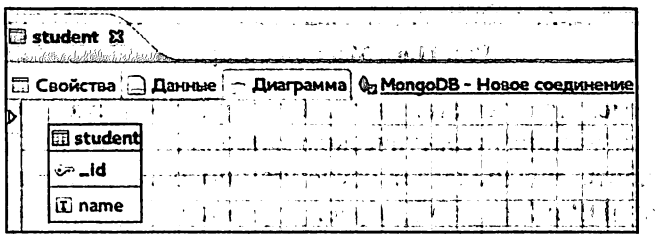


Рис. 11.58. Вкладка **Диаграмма** для СУБД MongoDB

Заметим, что в SQL базы данных информация также может быть занесена путем создания соответствующего скрипта.

Вопросы работы с пользователями и администрирование при помощи DBeaver предлагаются в качестве самостоятельного упражнения.

Для одновременной работы с таблицами различных баз данных достаточно просто открыть каждую из них в своей вкладке (рис. 11.59).

При этом в **Менеджере запросов** будет отражено открытие и таблицы **table1** СУБД MariaDB, и коллекции **student** СУБД MongoDB (рис. 11.60).

Таким образом, использование универсального менеджера баз данных Dbeaver дает пользователям возможность одинаково удобно работать как с SQL, так и с NoSQL СУБД.

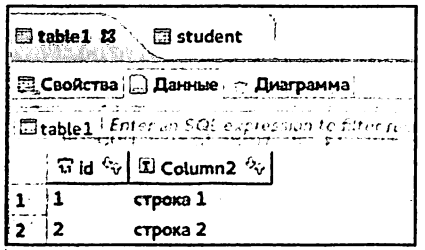


Рис. 11.59. Вкладки таблиц СУБД MariaDB и СУБД MongoDB

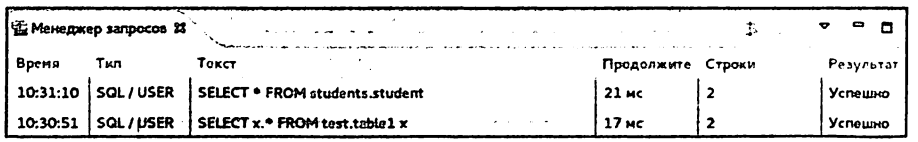


Рис. 11.60. Менеджер запросов

Учащимся предлагается выполнить практическую работу, основанную на изученном теоретическом материале. Выполнение практической работы предполагает изучение способов работы DBeaver с СУБД MariaDB и MongoDB, изучение работы с ними одновременно, создавая несколько соединений и открывая таблицы и коллекции для работы в различных вкладках одного окна.

Порядок выполнения практической работы

1. Установить и запустить DBeaver.
2. Создать соединение с СУБД MariaDB.
3. В соответствии с заданием создать базу данных и таблицы. Внести данные.
4. Проверить, каким образом созданные таблицы отображаются на диаграмме.
5. Создать соединение с СУБД MongoDB.
6. В соответствии с заданием создать базу данных и коллекции. Внести данные, используя скрипты.
7. Открыть вкладку **Диаграмма** и посмотреть, каким образом созданные коллекции в ней отображаются.
8. Проверить возможность работы одновременно с таблицами MariaDB и MongoDB.
9. Сохранить результаты выполнения и выйти.

Контрольные вопросы

1. Для чего служит DBeaver? С какими СУБД он работает? Какие функции он выполняет?
2. Каким образом создается новое соединение? Какие параметры необходимо задать?
3. Расскажите о процессе создания таблиц базы данных MariaDB.
4. Чем отличается процесс создания коллекций в СУБД MariaDB от процесса создания таблиц в СУБД MongoDB?
5. Расскажите, какие способы внесения данных в таблицы/коллекции вам известны. В чем особенность представления данных в СУБД MongoDB по сравнению с оболочкой Robomongo?
6. Какая информация отображается в окне **Менеджер запросов**?
7. Какая информация отображается во вкладке **Диаграмма**?
8. Каким образом осуществляется одновременная работа с таблицами различных СУБД?

Лабораторная работа 12
ПРИМЕР ИСПОЛЬЗОВАНИЯ SQL БАЗЫ ДАННЫХ MARIADB

Рассмотрим пример сайта, созданный с использованием СУБД MariaDB. Предполагается, что средства, используемые для создания интерфейса, такие как HTML5, CSS, JavaScript и библиотека jQuery, читателю известны. Кроме того, заметим, что для простоты и краткости изложения данные не будут приведены к третьей нормальной форме, поскольку это потребует создания нескольких дополнительных таблиц-справочников, что неизбежно приведет к необходимости их подробного описания и усложнению программного кода.

В качестве операционной системы, как и во всех предыдущих примерах, используется Linux Fedora. В качестве браузера — Mozilla Firefox. Если используются иные браузеры, то необходимо проверить, что изображение соответствует приведенной в тексте картинке, поскольку различные браузеры могут иметь особенности при работе с HTML5 и CSS3(4) [7, 8].

В настоящий момент средства разработки интерфейсов под Веб, например jQuery, обладают настолько универсальным инструментарием, что данная технология может быть применена для доступа к базам данных как в локальных сетях, так и через сеть Интернет. В данном пособии предполагается, что база данных установлена на сервере локальной сети. Преимущество такого подхода заключается в том, что на рабочих станциях клиенты получают доступ к данным через браузер, что обеспечивает кроссплатформенность и отсутствие затрат по установке специфических настроек.

Рассмотренный ниже пример не претендует на полноту охвата использования всевозможных элементов для создания графического

интерфейса, однако некоторые основные полезные приемы, которые используются практически повсеместно, будут рассмотрены подробно.

В качестве примера применения SQL базы данных MariaDB рассмотрим информационную систему. Предположим, что на некотором факультете имеется химическая и физическая лаборатории, а также компьютерный класс. В каждом из этих классов студенты в количестве не более шести человек имеют возможность выполнять лабораторные работы, для чего необходимо записаться заранее на определенный день. Предполагается, что время, когда лабораторные классы свободны, заранее известно, поскольку на факультетах стараются составить расписание таким образом, чтобы студенты могли пользоваться лабораториями вечером.

Таким образом, необходимо хранить данные о выбранном классе, студенте, его группе и примечания. Создадим, как было рассказано выше, для этого новую базу данных **lab** и в ней таблицы **students** (рис. 12.1) для хранения данных о записавшихся в лабораторный класс студентах и **typelab** — таблицу-справочник для хранения названия лабораторных классов (рис. 12.2).

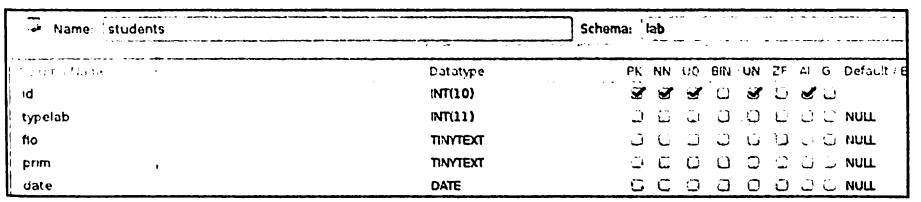


Рис. 12.1. Таблица **students**

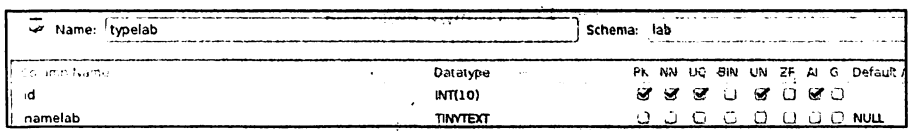
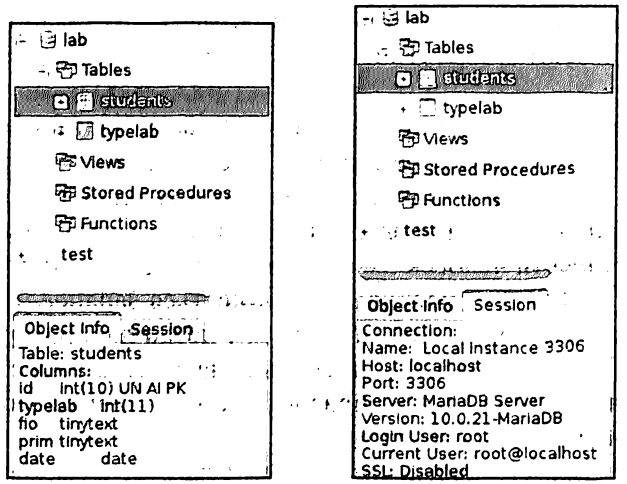


Рис. 12.2. Таблица **typelab**

После создания базы данных и таблиц информация о них появится во вкладке **SCHEMAS**, а внизу можно увидеть информацию (рис. 12.3) о выбранном объекте (вкладка Object Info, таблица **students**) и о сессии (вкладка **Session**, настройки хоста, порта и версии MariaDB).



а *б*

Рис. 12.3. Информация о выбранном объекте (*а*) и сессии (*б*)

Далее рассмотрим возможный интерфейс (рис. 12.4), в котором предусмотрены поля для выбора класса, внесения фамилии, имени и отчества студента, номера группы и выбора даты.

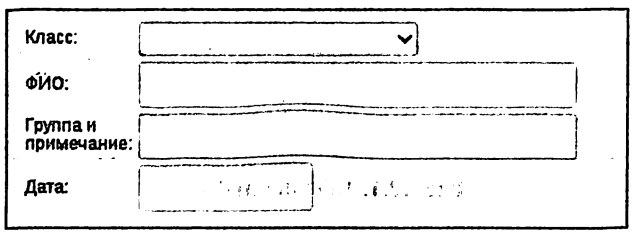


Рис. 12.4. Общий вид окна

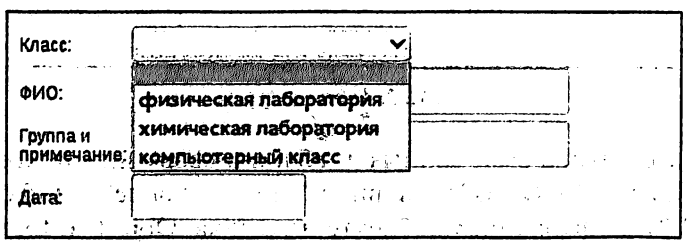


Рис. 12.5. Раскрывающийся список для выбора лабораторий

В окне будет один раскрывающийся список лабораторных классов (рис. 12.5).

Далее необходимо внести данные о студенте и выбрать дату, на которую осуществляется запись. Для этого нужно кликнуть мышкой по полю **Дата** и в открывшемся календаре выбрать день (рис. 12.6). Заметим, что в приведенном ниже коде не предусматривается блокировка записи на выходные (или праздничные дни), однако это легко сделать при помощи соответствующей проверки.

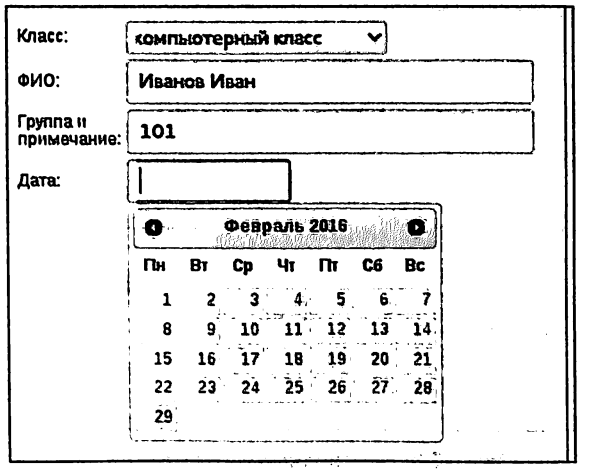


Рис. 12.6. Выбор даты

После того как данные внесены, на экране появляется кнопка **Записаться**. Нажимаем ее для внесения информации в базу, данные о студенте появляются справа от формы (рис. 12.7).

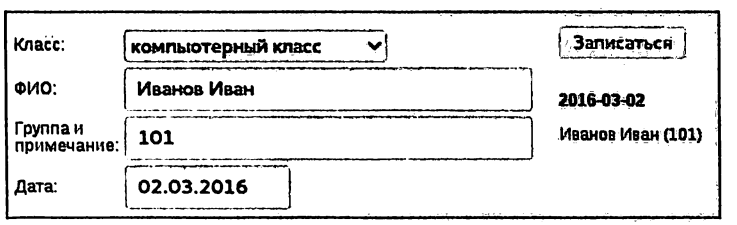


Рис. 12.7. Данные о студенте внесены

Если во время заполнения формы некоторые поля останутся пустыми, то в модальном окне появится предупреждение. Например,

если не будет заполнено поле **ФИО**, то во всплывающем окне появится предупреждение, как показано на рис. 12.8.

В том случае, когда оставлено пустым поле **Дата**, то в окне появится предупреждение, как показано на рис. 12.9, если не выбран класс, то появится сообщение «Выберите лабораторию!» (рис. 12.10).

Далее предположим, что на эту дату внесено пять человек. Поскольку разрешено записаться всего шестерым, то после того, как

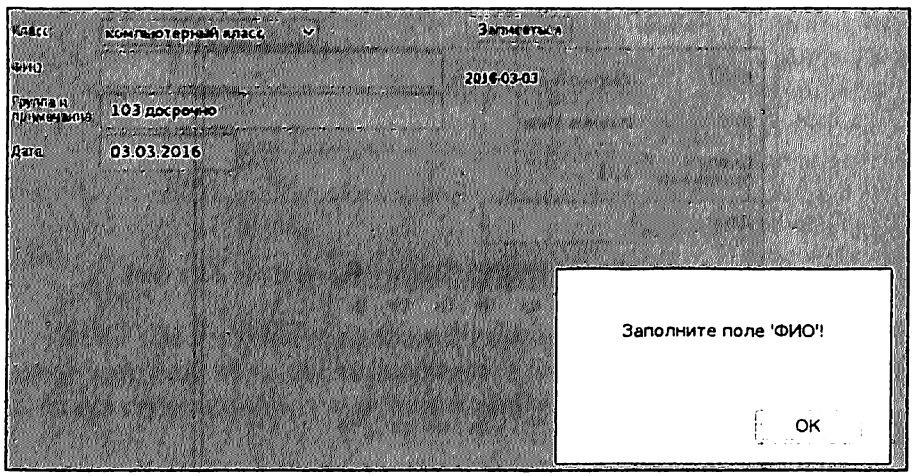


Рис. 12.8. Не заполнено поле **ФИО**

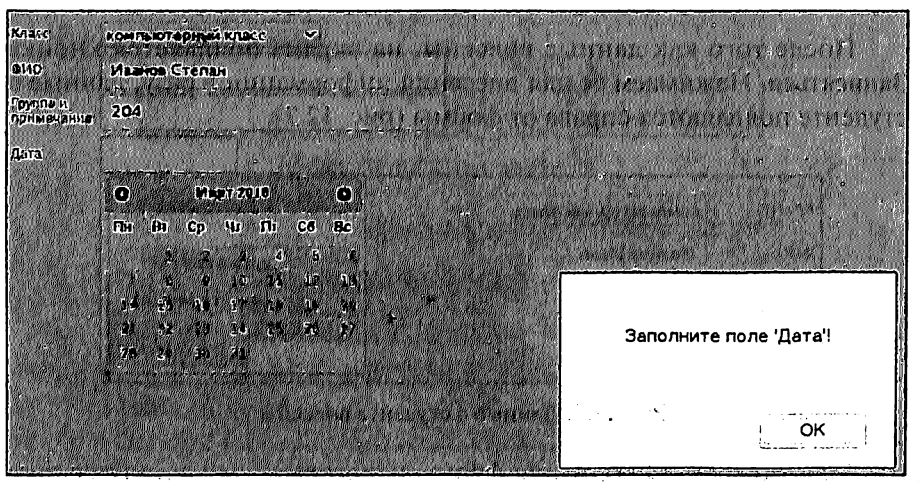


Рис. 12.9. Не заполнено поле **Дата**

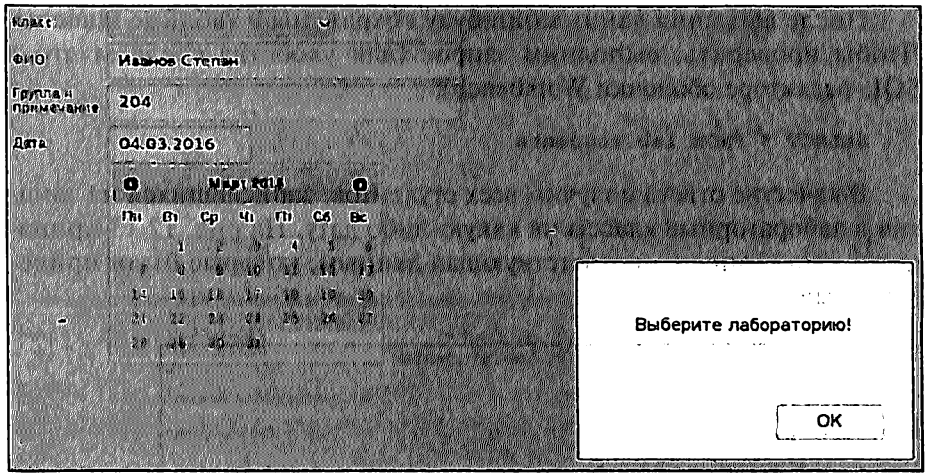


Рис. 12.10. Не заполнено поле **Класс**

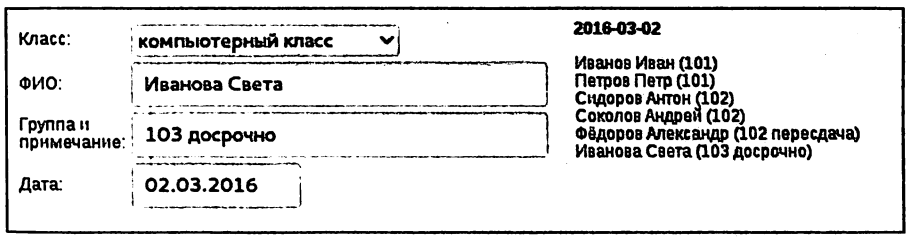


Рис. 12.11. Записаны разрешенные шесть человек

данные шестого человека будут внесены, кнопка **Записаться** не будет отображаться на эту дату (рис. 12.11).

При записи нового студента на другую дату в том случае, если на эту дату число записей меньше разрешенного количества, кнопка **Записаться** появится (рис. 12.12).

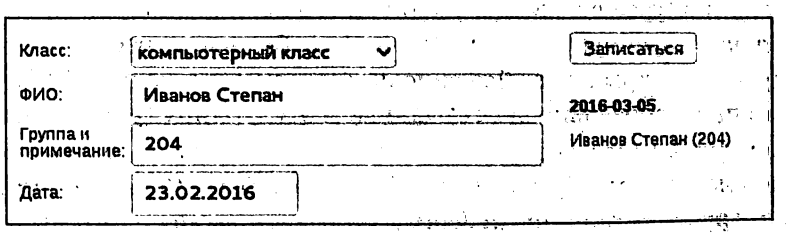
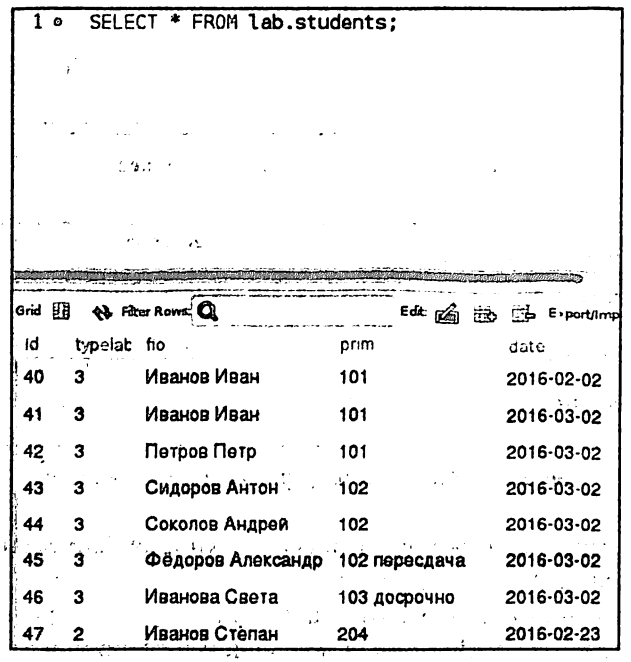


Рис. 12.12. Запись на другую дату

После внесения данных они буду сохранены в таблице **students**. Чтобы проверить, выполним запрос (для удобства сделаем это в SQL-редакторе оболочки Workbench):

```
SELECT * FROM lab.students
```

В качестве ответа получим всех студентов, записавшихся на занятия в лабораторные классы на какую-либо дату. На рис. 12.13 приводится часть записи, соответствующая данным, которые были приведены выше.



```
1 ⚬ SELECT * FROM lab.students;
```

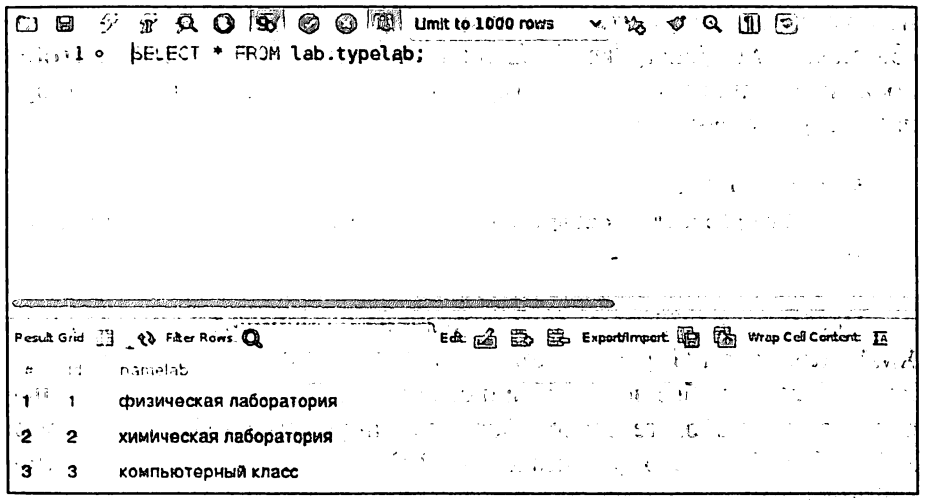
Grid | Filter Rows: | Edit: | Export/Imp

id	typelab	fio	prim	date
40	3	Иванов Иван	101	2016-02-02
41	3	Иванов Иван	101	2016-03-02
42	3	Петров Петр	101	2016-03-02
43	3	Сидоров Антон	102	2016-03-02
44	3	Соколов Андрей	102	2016-03-02
45	3	Фёдоров Александр	102 пересдача	2016-03-02
46	3	Иванова Света	103 досрочно	2016-03-02
47	2	Иванов Степан	204	2016-02-23

Рис. 12.13. Содержимое таблицы **students**

Таблица **typelab** является таблицей-справочником и содержит названия лабораторных классов. Для того чтобы начать работу с формой и получить раскрывающийся список, как показано на рис. 12.5, необходимо предварительно внести данные в эту таблицу (рис. 12.14).

Еще раз оговоримся, что данные в базе не находятся в третьей нормальной форме. Для этого необходимо добавление полей **имя**, **отчество**, **группа** в таблицу **students** и соответствующих таблиц в базу. Хотя указанное добавление увеличит объем описания и программного кода, однако принципиально оно не повлияет на функционирование системы, поэтому в данном учебном проекте не будем добавлять указанные таблицы в базу.



```
1 ⚬ SELECT * FROM lab.typelab;
```

Limit to 1000 rows

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

#	id	namelab
1	1	физическая лаборатория
2	2	химическая лаборатория
3	3	компьютерный класс

Рис. 12.14. Данные таблицы **typelab**

Ниже приведен программный код, состоящий из нескольких файлов. Основным вызываемым файлом является **lab.htm**. В этом файле сначала подключим необходимые файлы для работы с библиотеками jQuery и календарем из библиотеки jQuery, команды:

```
<link rel="stylesheet"
href="//code.jquery.com/ui/1.11.4/themes/smoothness/jquery-ui.
css">
<script src="//code.jquery.com/jquery-1.10.2.js"></script>
<script
src="//code.jquery.com/ui/1.11.4/jquery-ui.js"></script>
<script src="datepicker-ru.js"></script>
```

Функция **MyOutCell** служит для проверки правильности заполнения полей и отправки асинхронным методом Ajax данных полей на сервер.

Виджет **DatePicker** позволяет пользователям выбирать необходимую дату. С помощью метода **datepicker** можно превратить выбранный элемент (например, текстовое поле) в виджет. Программный код файла **datepicker-ru.js** приведен ниже. Заметим, что он предварительно скачивается с сайта библиотеки jQuery UI (http://jqueryui.com /download). Для того чтобы можно было использовать русский язык,

необходимо вставить ссылку на скрипт локализации (**$.datepicker.regional["ru"]**). Для вызова виджета **DatePicker**, чтобы добавлять к простому текстовому полю формы календарь для выбора даты, служит функция

```
$(function() {
$( "#datepicker" ).datepicker( $.datepicker.regional[ "ru" ] );
});
```

Необходимо также обратить внимание на использование объекта **XMLHttpRequest** [11]. Этот объект позволяет осуществлять HTTP-запросы к серверу без перезагрузки страницы. Информация может передаваться в любом текстовом формате, например в XML, HTML или JSON. Благодаря этому технология AJAX работает в сочетании с новейшими API.

Принцип его работы:

- создается экземпляр встроенного объекта **XMLHttpRequest** при помощи команды var xhr = new XMLHttpRequest() для отправки запроса на сервер;
- объект **xhr** конфигурируется, используемый метод — **POST**;
- далее вызывается обработчик события **xhr.onload** и проверяется HTTP-статус **this.status** (статус имеет целочисленное значение, например 404 — «Not Found», 200 — «OK», 500 — «Internal Server Error»). Если **this.status** не равен 200, то выдается сообщение об ошибке при помощи метода **alert**;
- если для дальнейшей работы необходимо дождаться завершения передачи данных, то необходимо проверять **this.readyState**, который возвращает статус текущего документа. Значение изменяется от 0 до 4 в процессе выполнения запроса (0 — не инициализирован; при инициализации запроса — 1, при отправке — 2, при частичном получении ответа — 3, при завершении выполнения — 4. Поэтому проверка осуществляется по двум условиям: **this.readyState == 4 && this.status == 200**;
- если обмен успешный, то ответ **this.response** будет содержать данные, полученные от сервера;
- для отправки данных форм используется метод **send**: **xhr.send(FormData)**. Тип FormData — тип данных в рамках технологии XHR2 для динамического создания HTML-элементов <form> с помощью JavaScript.

Код основного файла **lab.htm** (код):

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Запись студентов в лабораторию</title>

<link rel="stylesheet"
href="//code.jquery.com/ui/1.11.4/themes/smoothness/jquery-ui.
css">
<script src="//code.jquery.com/jquery-1.10.2.js"></script>
<script src="//code.jquery.com/ui/1.11.4/jquery-ui.js"></script>
<script src="datepicker-ru.js"></script>

<script>
/* ********************************************************** */
MyOutCell = function(nSignAdd)
{
var sAux, aAux;
var f = document.getElementById("formstudent");
var formData = new FormData();

//
sAux = f.elements["typelab"].value;
sAux = sAux.trim();
  if(parseInt(sAux, 10) === 0) { alert("Выберите
лабораторию!"); return false; }
formData.append("typelab", sAux);
//
sAux = f.elements["fio"].value;
sAux = sAux.trim();
  if((sAux === '') && (nSignAdd == 1)) { alert("Заполните поле
'ФИО'!"); return false; }
formData.append("fio", sAux);
//
sAux = f.elements["prim"].value;
sAux = sAux.trim();
  if((sAux === '') && (nSignAdd == 1)) { alert("Заполните поле
'Примечание'!"); return false; }
formData.append("prim", sAux);
//
sAux = f.elements["datepicker"].value;
sAux = sAux.trim();
  if(sAux === '') { alert("Заполните поле 'Дата'!"); return
false; }
aAux = sAux.split('.');
sAux = aAux[2]+"-"+aAux[1]+"-"+aAux[0];
formData.append("date", sAux);
```

```
//
formData.append("signadd", nSignAdd);
//

var xhr = new XMLHttpRequest();
xhr.open("POST", "outcell.php");
xhr.onload = function(e)
{
  if(this.readyState == 4 && this.status == 200)
  {
  sAux = this.response;
  sAux = sAux.trim();
  aAux = sAux.split("#");
    if(parseInt(aAux[0], 10) === 0) { alert(aAux[1]);
    return false; }
    else { sAux = aAux[1]; }
  sAux = sAux.trim();
  document.getElementById('outcell').innerHTML = sAux;
  }
  else
  {
  alert("Err!"); return false;
  }
return false;
};
xhr.send(formData);
return false;
};

/* ******************************************************* */
$(function() {
  $( "#datepicker" ).datepicker( $.datepicker.regional[ "ru" ] );
});

</script>

<style type='text/css'>

html, body {
    height: 100%
}

body {
  padding: 1px 0;
  background-color: #f1f1f1;
  font-family:arial;font-size:12px;color:#000;
}

td {
  text-align: left;
```



```
  vertical-align: middle;
}
</style>
</head>

<body>
<?php
include ("conn.php");
?>
<form id=formstudent>
<table>
<tr>
  <td style="vertical-align: top;">
  <table>
    <tr><td>Класс:</td><td>
    <select id="typelab"
onChange="document.getElementById('outcell').innerHTML = '';">
<?php
$sOut = "<OPTION VALUE=\"0\"></OPTION>\n";
$result = $pdo->prepare('SELECT * FROM typelab;');
$result->execute();
if($result->rowCount() != 0) {
  foreach($result as $row) {
    $id = "{$row['id']}";
    $namelab = "{$row['namelab']}";
    $sOut .= "<OPTION VALUE=\"$id\">$namelab</OPTION>\n";
  }
}
echo $sOut;
?>
    </select>
    </td></tr>
    <tr><td>ФИО:</td><td><input type="text" id="fio"
size=30></td></tr>
    <tr><td>Группа и<br>примечение:</td><td><input type="text"
    id="prim" size=30></td></tr>
    <tr><td>Дата:</td><td><input type="text" id="datepicker"
    size=10 onChange="MyOutCell(0); return false;"></td></tr>
  </table>
  </td>
  <td>   </td>
  <td style="vertical-align: top;" id=outcell></td>
</tr>
</table>
</form>

</body>
</html>
```

Код файла виждета **datepicker-ru.js**:

```
/* Russian (UTF-8) initialisation for the jQuery UI date
picker plugin. */
/* Written by Andrew Stromnov (stromnov@gmail.com). */
( function( factory ) {
    if ( typeof define === "function" && define.amd ) {

        // AMD. Register as an anonymous module.
        define( [ "../widgets/datepicker" ], factory );
    } else {

        // Browser globals
        factory( jQuery.datepicker );
    }
}( function( datepicker ) {

datepicker.regional.ru = {
    closeText: "Закрыть",
    prevText: "&#x3C;Пред",
    nextText: "След&#x3E;",
    currentText: "Сегодня",
    monthNames: [
"Январь","Февраль","Март","Апрель","Май","Июнь",
    "Июль","Август","Сентябрь","Октябрь","Ноябрь","Декабрь" ],
    monthNamesShort: [ "Янв","Фев","Мар","Апр","Май","Июн",
    "Июл","Авг","Сен","Окт","Ноя","Дек" ],
    dayNames: [
    "воскресенье","понедельник","вторник","среда","четверг",
    "пятница","суббота" ],
    dayNamesShort: [ "вск","пнд","втр","срд","чтв","птн","сбт" ],
    dayNamesMin: [ "Вс","Пн","Вт","Ср","Чт","Пт","Сб" ],
    weekHeader: "Нед",
    dateFormat: "dd.mm.yy",
    firstDay: 1,
    isRTL: false,
    showMonthAfterYear: false,
    yearSuffix: "" };
datepicker.setDefaults( datepicker.regional.ru );

return datepicker.regional.ru;

} ) );
```


Установка связи с базой данных происходит в файле **conn.php**. Для соединения используется PDO (PHP Data Objects) — расширение для PHP (доступно, начиная с версии 5.1), которое предоставляет разработчику универсальный интерфейс для доступа к различным СУБД, однако для каждого типа СУБД следует установить свой драйвер доступа.

Обратите внимание, что для установки способа обработки ошибок используется атрибут **PDO::ATTR_ERRMODE**. Для определения, какая строка будет возвращаться в вызывающий метод, служит атрибут **PDO::ATTR_CURSOR**. Для задания типа получаемого результата по умолчанию (в каком виде следующая строка будет возвращена в вызывающий метод) используется атрибут **PDO::ATTR_DEFAULT_FETCH_MODE**.

Код файла **conn.php**:

```
<?php
$host = "127.0.0.1";
$db = "lab";
$charset = "utf8";
$dsn = "mysql:host=$host;dbname=$db;charset=$charset";
$opt = array(
   PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
   PDO::ATTR_CURSOR => PDO::CURSOR_FWDONLY,
   PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
);
$user = "labuser";
$pass = "labpass";
$pdo = new PDO($dsn, $user, $pass, $opt);
```

Для работы непосредственно с базой данных, т. е. занесения информации в базу и получения информации из базы, служит файл **outcell.php**. Обратите внимание на функцию **trim()**, которая позволяет убрать лишние пробельные символы (пробелы, табуляцию и т. п.) из начала и конца строки. Также следует обратить внимание, что для передачи данных используется метод **POST**. В этом случае (в отличие от метода **GET**) все названия переменных и значения будут передаваться как запрос браузера к веб-серверу, т. е. не будут видны в адресной строке. Существует суперглобальный массив **$_POST**, из которого можно считывать данные следующим образом: **$_POST['имя_переменной']**.

Код файла **outcell.php**:

```
<?php
header('Content-Type: text/html; charset=utf-8');
$sOut = "";
//------------------------------------------------------------
include ("conn.php");
//------------------------------------------------------------
$signadd = trim($_POST['signadd']);
//------------------------------------------------------------
$typelab = trim($_POST['typelab']);
  if($typelab == "") {
   echo "0#Ошибка! Заполните поле 'Класс'!"; die();
  }
//------------------------------------------------------------
$fio = trim($_POST['fio']);
  if(($fio == "") && ($signadd == 1)) {
   echo "0#Ошибка! Заполните поле 'ФИО'!"; die();
  }
//------------------------------------------------------------
$prim = trim($_POST['prim']);
  if(($prim == "") && ($signadd == 1)) {
   echo "0#Ошибка! Заполните поле 'Примечание'!"; die();
  }
//------------------------------------------------------------
$date = trim($_POST['date']);
  if($date == "") {
   echo "0#Ошибка! Заполните поле 'Дата'!"; die();
  }
//------------------------------------------------------------
if($signadd == 1) {
$result = $pdo->prepare('INSERT INTO students (typelab, fio,
prim, date) VALUES (?,?,?,?);');
$result->execute([$typelab,$fio,$prim,$date]);
}
//------------------------------------------------------------
$result = $pdo->prepare('SELECT * FROM students WHERE
typelab=? AND date=?;');
$result->execute([$typelab,$date]);
if($result->rowCount() <= 5)
{
$sOut .= "<button onClick=\"MyOutCell(1); return
false;\">Записаться</button><br><br><br>";
}
```



```
$sOut .= "<b>$date</b><br><br>";
if($result->rowCount() != 0) {
  foreach($result as $row) {
   $fio = "{$row['fio']}";
   $prim = "{$row['prim']}";
   $sOut .= "$fio ($prim)<br>";
  }
}
//------------------------------------------------------------
echo "1#".$sOut;
//------------------------------------------------------------
?>
```

Контрольные вопросы

1. Для чего служит виджет **DatePicker** из библиотеки jQuery? Каким образом он вызывается?
2. Какие дополнительные действия следует произвести, чтобы для виджета **DatePicker** настроить русский язык?
3. Для каких целей служит функция **MyOutCell**?
4. В каких случаях используется объект **XMLHttpRequest**?
5. Что такое **PDO PHP**? Какие атрибуты были заданы и для чего?
6. Какая кодировка данных используется?
7. Для чего служит функция **trim()**?
8. Почему данные передаются методом **POST**?

Лабораторная работа 13
ПРИМЕР ИСПОЛЬЗОВАНИЯ NOSQL БАЗЫ ДАННЫХ MONGODB

Пример, рассматриваемый ниже, как и пример для СУБД MariaDB, реализован на операционной системе Linux Fedora и браузере Mozilla Firefox. Следует помнить, что при использовании иных браузеров не все из них в равной степени поддерживают возможности HTML5 и CSS3(4) [7,8].

В качестве примера применения NoSQL базы данных MongoDB возьмем информационную систему. Предположим, что необходимо хранить данные о преподаваемых предметах, темах, которые необходимо изучить, и контрольные вопросы к ним.

Сначала рассмотрим возможный интерфейс (рис. 13.1). В окне имеется два раскрывающихся списка: для предметов (вверху) и тем, соответствующих предмету (в нижней части). Кроме того, необходимо создать поле для отображения контрольных вопросов. Поскольку их может быть много, целесообразно отображать их вне таблицы. Очевидно, что необходимо иметь возможность добавлять (знак «+»), изменять (знак «/»), удалять (знак «–») предмет (или дисциплину) и отображать служебную информацию (знак «?», что особенно актуально при администрировании системы).

Рис. 13.1. Примерный интерфейс информационной системы

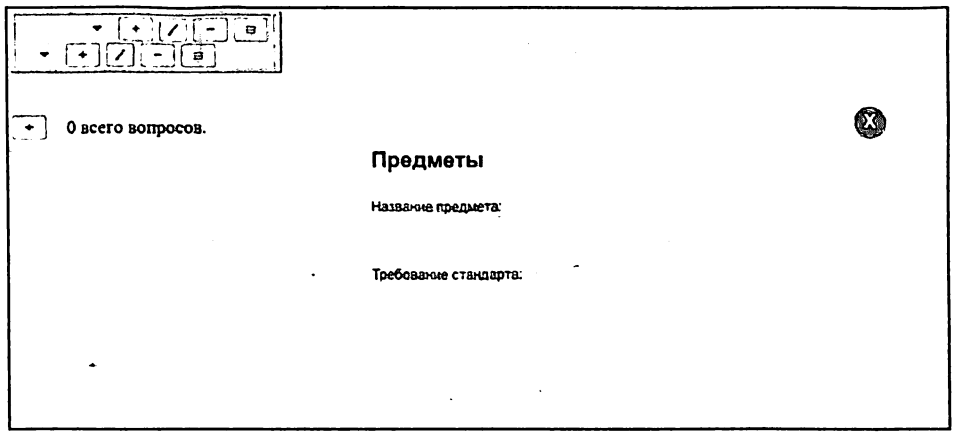


Рис. 13.2. Окно для ввода информации о предмете

Внесем предмет, например «Математика». При нажатии на кнопку со знаком «+» откроется окно (рис. 13.2). Внесем в него данные, как показано на рис. 13.3. Результат выполнения представлен на рис. 13.4.

В случае необходимости администрирования имеется возможность отображения в окне служебной информации (рис. 13.5). Весь программный код с подробными комментариями описан ниже.

Далее аналогичным образом внесите данные о темах. Для этого в верхнем раскрывающемся списке выберите требуемый предмет (в данном случае «Математика») и, как и ранее, при нажатии на кнопку со знаком «+» откроется окно (рис. 13.6). После внесения данных

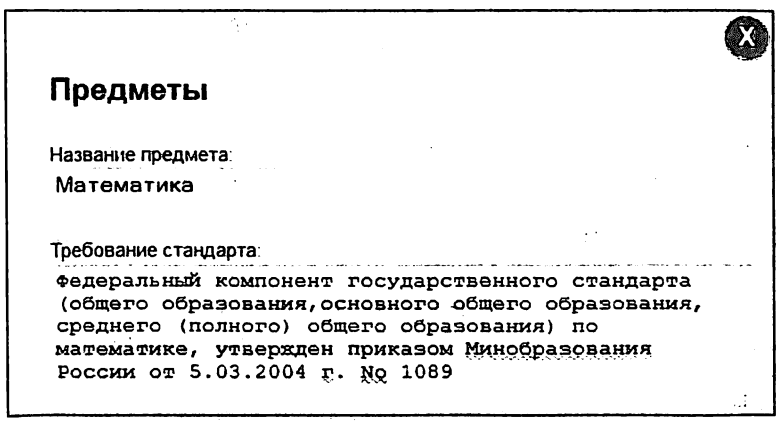


Рис. 13.3. Заполнение данных о предмете

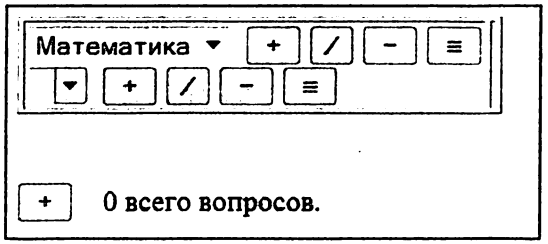


Рис. 13.4. Данные о предмете внесены

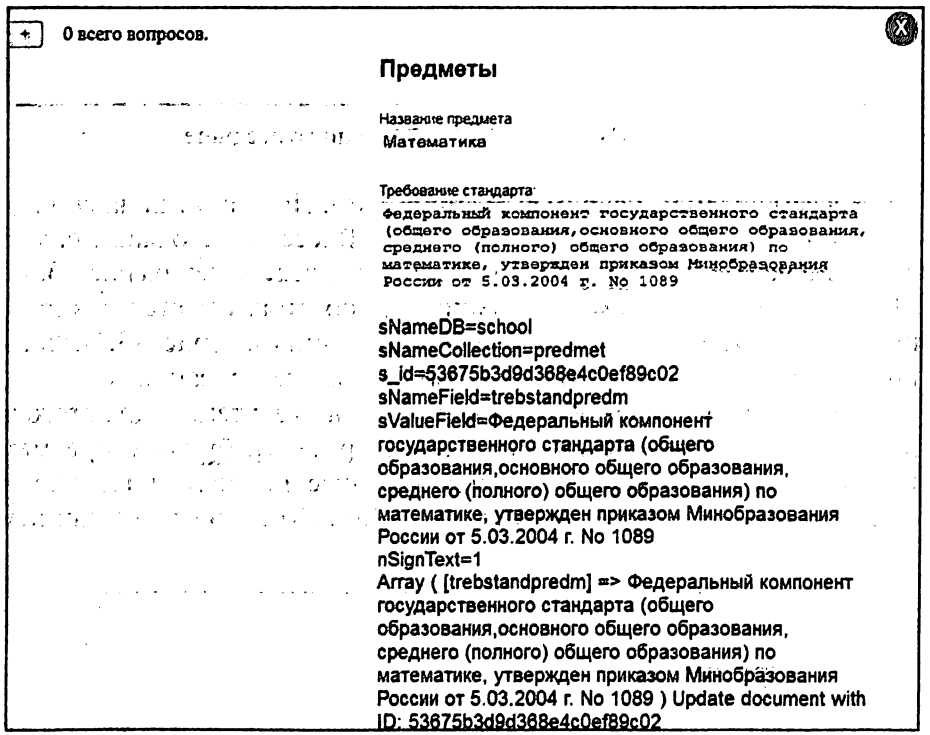


Рис. 13.5. Отображение служебной информации

(рис. 13.7) результат выполнения будет отображен аналогично представленному на рис. 13.7.

Для внесения вопросов нажмите на кнопку со знаком «+» в поле для внесения вопросов (рис. 13.8). Результат заполнения формы представлен на рис. 13.9. Обратите внимание: после внесения вопросов они сортируются по порядку, т. е. если, например, вопрос номер 2 внести позже или перенумеровать вопросы в процессе редактирования, окончательное расположение вопроса все равно будет соответствовать его порядковому номеру (рис. 13.10).

Для вопросов также предусмотрена возможность редактирования (знак «/») и удаления (знак «—»). Для внесения вопросов, соответствующих конкретной теме, из списка тем выберите требуемую и последовательно внесите вопросы (рис. 13.11).

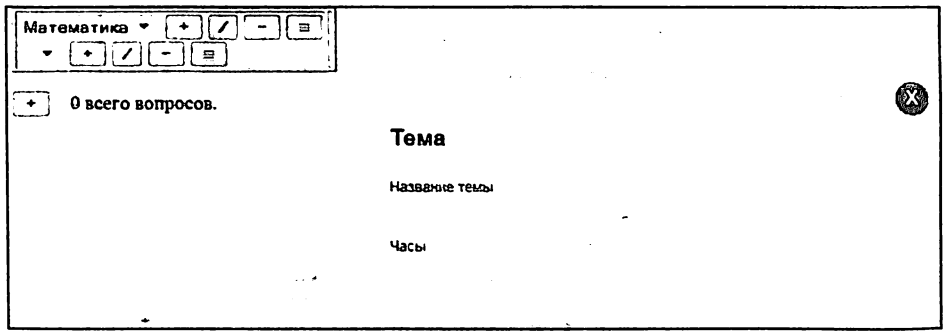


Рис. 13.6. Окно для ввода информации о теме

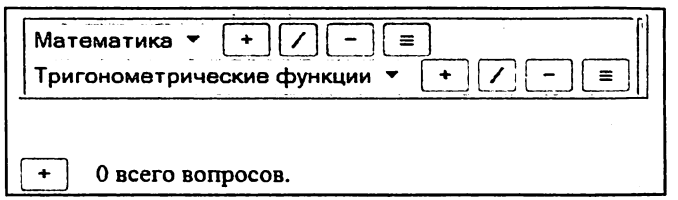


Рис. 13.7. Данные о теме внесены

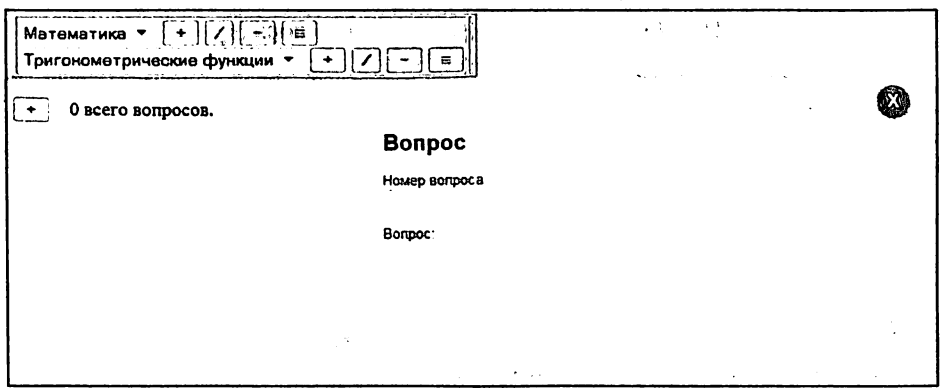


Рис. 13.8. Окно для ввода вопросов

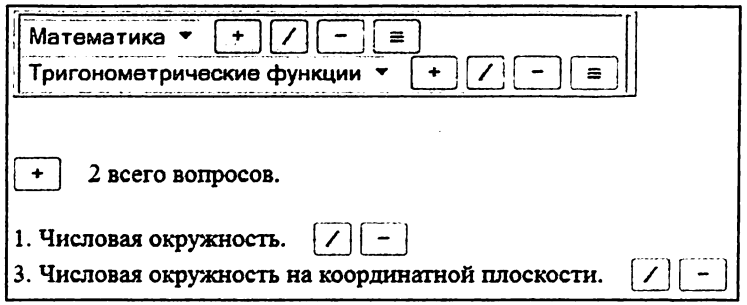


Рис. 13.9. Вопросы внесены

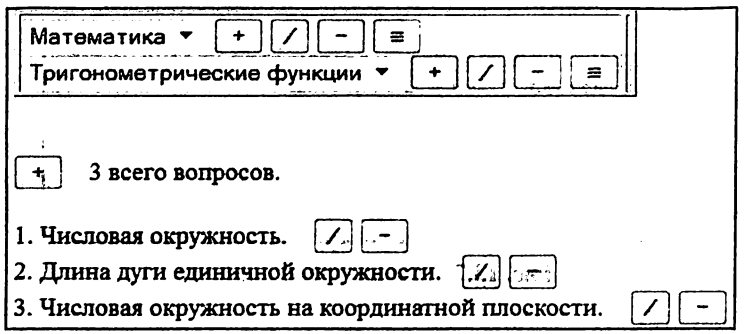


Рис. 13.10. Вопросы сортируются по порядку

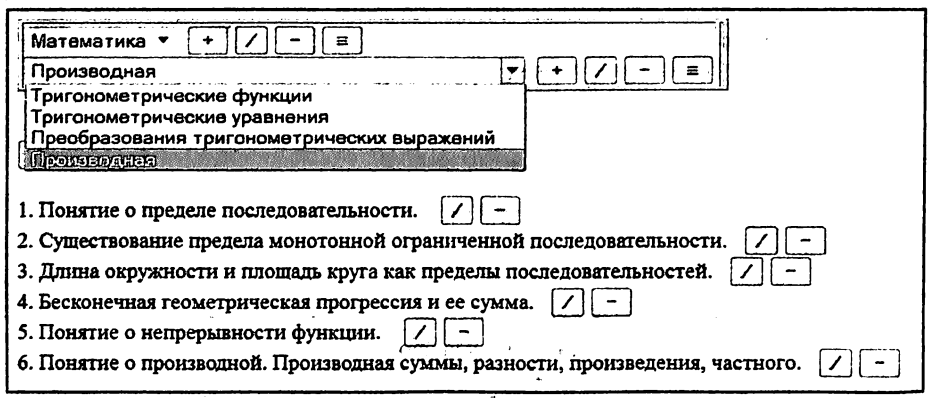


Рис. 13.11. Внесение вопросов, соответствующих выбранной теме

После внесения всех вопросов по теме она приобретает вид, представленный на рис. 13.12.

В случае необходимости получения служебной информации о внесенных данных о предметах (темах) для отладки или администрирования следует нажать на знак «≡». Результаты выполнения данной операции представлены на рис. 13.13 и 13.14 соответственно.

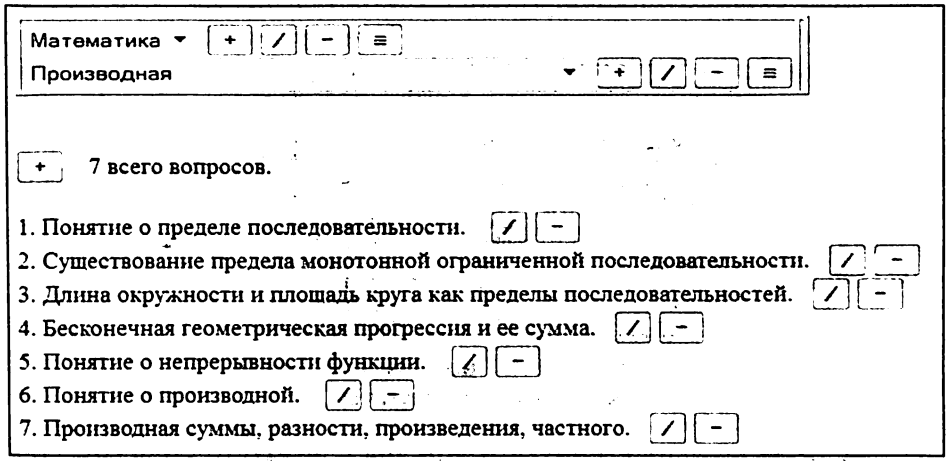


Рис. 13.12. Окончательный вид внесенных вопросов по теме

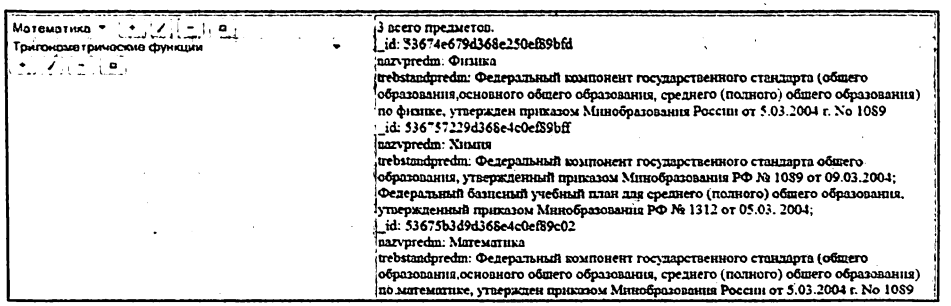


Рис. 13.13. Служебная информация о внесенных предметах

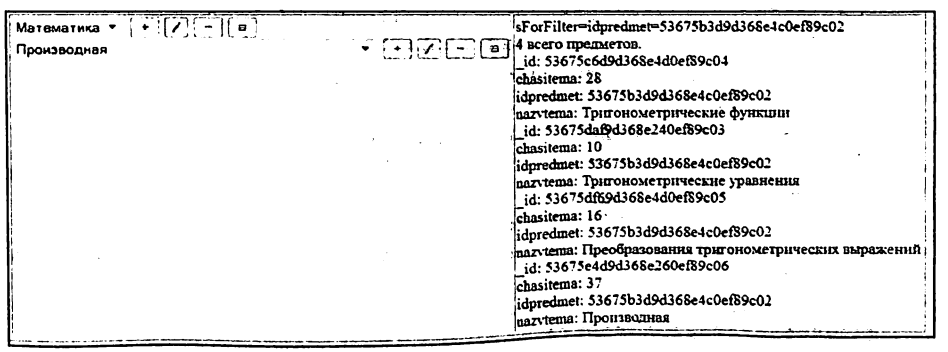
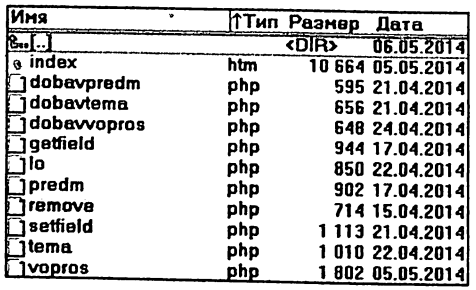


Рис. 13.14. Служебная информации о внесенных темах

Далее рассмотрим программный код, реализующий взятую для примера информационную систему. На рис. 13.15 представлены все необходимые для функционирования системы файлы.



Имя	Тип	Размер	Дата
[..]		<DIR>	06.05.2014
index	htm	10 664	05.05.2014
dobavpredm	php	595	21.04.2014
dobavtema	php	656	21.04.2014
dobavvopros	php	648	24.04.2014
getfield	php	944	17.04.2014
lo	php	850	22.04.2014
predm	php	902	17.04.2014
remove	php	714	15.04.2014
setfield	php	1 113	21.04.2014
tema	php	1 010	22.04.2014
vopros	php	1 802	05.05.2014

Рис. 13.15. Файлы, необходимые для функционирования системы

Как было сказано выше, объединение коллекций производится при помощи сохранения поля **_id** одного документа в другом документе в качестве ссылки. Рассмотрим, например, содержимое коллекции «Тема». Обратите внимание, что ссылки на предмет (**_id** предмета) в темах, относящихся к одному предмету, совпадают:

```
_id: 53675c6d9d368e4d0ef89c04 chasitema: 28 idpredmet:
53675b3d9d368e4c0ef89c02 nazvtema: Тригонометрические функции
_id: 53675daf9d368e240ef89c03 chasitema: 10 idpredmet:
53675b3d9d368e4c0ef89c02 nazvtema: Тригонометрические
уравнения
_id: 53675df69d368e4d0ef89c05 chasitema: 16 idpredmet:
53675b3d9d368e4c0ef89c02 nazvtema: Преобразования
тригонометрических выражений
_id: 53675e4d9d368e260ef89c06 chasitema: 37 idpredmet:
53675b3d9d368e4c0ef89c02 nazvtema: Производная
_id: 536755d59d368e4d0ef89c00 chasitema: 32 idpredmet:
53674e679d368e250ef89bfd nazvtema: Механика
_id: 536756449d368e280ef89bff chasitema: 32 idpredmet:
53674e679d368e250ef89bfd nazvtema: Молекулярная физика.
Термодинамика
_id: 5367574f9d368e4d0ef89c03 chasitema: 8 idpredmet:
536757229d368e4c0ef89bff nazvtema: Углеводороды и их природные
источники
_id: 536758979d368e4c0ef89c00 chasitema: 10 idpredmet:
536757229d368e4c0ef89bff nazvtema: Кислородсодержащие
органические соединения и их природные источники
```


Рассмотрим более подробно программный код и функционирование информационной системы в целом. Начнем с основного файла **index.htm**.

Вначале при помощи HTML5 и CSS3 создается модальное окно. Для этого создайте класс **modalDialog** и его стили. Положение окна задайте фиксированным (**position: fixed**), чтобы оно оставалось неподвижным при прокрутке; задайте фон окна, свойство **opacity** (напомним, что при значении, равном нулю, элемент становится полностью прозрачным, а при значении, равном единице, соответственно совсем непрозрачным), свойство **z-index** (порядок слоев; обратите внимание, что работает только для элементов, у которых значение **position** задано как **absolute**, **fixed** или **relative**) и свойства **CSS transitions**, которые позволяют производить изменения свойств CSS плавно в течение некоторого времени (подробнее о свойствах CSS3 см. [9]). В завершение переведите окно в неактивное состояние (т. е. **pointer-events: none**). Свойство **pointer-events** (формально относится к CSS4, но многие современные браузеры его поддерживают) позволяет определить, каким образом элементы будут реагировать на нажатие кнопки мыши. Если **pointer-events** установлен в **none**, то элемент не реагирует на такие события мыши, как **hover**, **click** и др., а события будут передаваться элементу, лежащему под ним.

Добавьте псевдокласс **target**, который служит для вывода элемента, и стили для модального окна внутри элемента **div**: определите ширину модального окна, положение на странице, цвет фона и т. п.

Создайте кнопку для закрытия окна (**close**), позиционируйте ее, задайте фон и спецэффекты (тень). При наведении курсора мыши на кнопку кнопка будет менять цвет фона **close:hover**.

В данном примере, так же как и в примере с СУБД MariaDB, используется объект **XMLHttpRequest** [11], позволяющий осуществлять HTTP-запросы к серверу без перезагрузки страницы. Напоминаем, что информация может передаваться в любом текстовом формате, например в XML, HTML или JSON. Благодаря этому технология AJAX работает в сочетании с новейшими API.

Код файла **index.htm**:

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
  <title>Учебный план</title>
  <style>
```

```
/*Определение стилей, в том числе модального окна*/
 .modalDialog {
   position: fixed;
   font-family: Arial, Helvetica, sans-serif;
   top: 0;
   right: 0;
   bottom: 0;
   left: 0;
   background: rgba(0,0,0,0.08);
   z-index: 99999;
   opacity:0;
   -webkit-transition: opacity 4ms ease-in;
   -moz-transition: opacity 4ms ease-in;
   transition: opacity 4ms ease-in;
   pointer-events: none;
 }

 .modalDialog:target {
   opacity:1;
   pointer-events: auto;
 }

 .modalDialog > div {
   width: 400px;
   position: relative;
   margin: 10% auto;
   padding: 5px 20px 13px 20px;
   border-radius: 10px;
   background: #fff;

 }

 .close {
   background: #606061;
   color: #FFFFFF;
   line-height: 25px;
   position: absolute;
   right: -12px;
   text-align: center;
   top: -10px;
   width: 24px;
   text-decoration: none;
   font-weight: bold;
   -webkit-border-radius: 12px;
   -moz-border-radius: 12px;
   border-radius: 12px;
   -moz-box-shadow: 1px 1px 3px #000;
   -webkit-box-shadow: 1px 1px 3px #000;
   box-shadow: 1px 1px 3px #000;
 }
```



```
 .close:hover { background: #ccccff; }

 .label1 { font-size:12px; }

 td {
 /* <http://www.w3.org/wiki/CSS/Properties/text-align>
 * left, right, center, justify, inherit
 */
 text-align: left;
 /* <http://www.w3.org/wiki/CSS/Properties/vertical-align>
 * baseline, sub, super, top, text-top, middle,
 * bottom, text-bottom, length, or a value in percentage
 */
 vertical-align: text-top;
 }

 </style>

<script>
/*Вводим глобальные переменные - _id соответствующей
коллекции*/
var _id_Predmet;
var _id_Tema;
var _id_Vopros;

/* ********************************************************** */
// Получение значение поля из заданной коллекции и документа с
указанным _id
// из заданной базы данных
function GetField(sNameDB, sNameCollection, s_id, sNameField,
sID)
{
var formData = new FormData();
formData.append("namedb", sNameDB);
formData.append("namecollection", sNameCollection);
formData.append("_id", s_id);
formData.append("namefield", sNameField);

var xhr = new XMLHttpRequest();
xhr.open("POST", "getfield.php");
xhr.onload = function(e)
{
  if(this.status == 200)
  {
  document.getElementById(sID).value = this.response;
  }
  else
  {
  alert("Err!");
  }
```

```
};
xhr.send(formData);
}

/* ********************************************************* */
// Установка значения поля в документ с указанным _id из
заданной коллекции
// в заданной базе данных
function SetField(sNameDB, sNameCollection, s_id, sNameField,
sValueField, nSignText)
{
var formData = new FormData();
formData.append("namedb", sNameDB);
formData.append("namecollection", sNameCollection);
formData.append("_id", s_id);
formData.append("namefield", sNameField);
formData.append("valuefield", sValueField);
formData.append("signtext", nSignText);

var xhr = new XMLHttpRequest();
xhr.open("POST", "setfield.php");
xhr.onload = function(e)
{
  if(this.status == 200)
  {
  document.getElementById("div1").innerHTML = this.response;
  }
  else
  {
  alert("Err!");
  }
};
xhr.send(formData);

}

/* ********************************************************* */
// Загрузка содержимого оператора Select
function loadOption(sNameDB, sNameCollection, sIDSelect,
sSufNazv, sFilter)
{
var formData = new FormData();
formData.append("namedb", sNameDB);
formData.append("namecollection", sNameCollection);
formData.append("sufnazv", sSufNazv);
formData.append("filter", sFilter);
```



```
var xhr = new XMLHttpRequest();
xhr.open("POST", "lo.php");
xhr.onload = function(e)
{
  if(this.readyState == 4 && this.status == 200)
  {
  document.getElementById(sIDSelect).innerHTML =
this.response;
    if(sNameCollection == 'predmet')
    {
    loadOption('school', 'tema', 'tema', 'tema',
'idpredmet='+predmet.value);
    }
    if(sNameCollection == 'tema')
    {
    //
    loadFileToDiv('vopros.php', 'divvopros',
'idtema='+tema.value);
    }
  }
  else
  {
  alert("Err!");
  }
};
xhr.send(formData);
}

/* ********************************************************* */
// Добавление предмета в коллекцию predmet в базе данных
function AddPredm()
{
var xhr = new XMLHttpRequest();
xhr.open("POST", 'dobavpredm.php');
xhr.onload = function(e)
{
  if(this.readyState == 4 && this.status == 200)
  {
  _id_Predmet = this.response;
  document.location.href='#openPredmet';
  }
  else
  {
  alert("Err!");
  }
};
xhr.send();
}
```

```
/* ********************************************************** */
//   Добавление темы в коллекцию tema в базе данных
function AddTema()
{
var formData = new FormData();
formData.append("idpredmet", _id_Predmet);

var xhr = new XMLHttpRequest();
xhr.open("POST", 'dobavtema.php');
xhr.onload = function(e)
{
  if(this.readyState == 4 && this.status == 200)
  {
  _id_Tema = this.response;
  document.location.href='#openTema';
  }
  else
  {
  alert("Err!");
  }
};
xhr.send(formData);
}

/* ********************************************************** */
//   Добавление вопроса в соответствующую коллекцию в базе
данных
function AddVopros()
{
var formData = new FormData();
formData.append("idtema", _id_Tema);

var xhr = new XMLHttpRequest();
xhr.open("POST", 'dobavvopros.php');
xhr.onload = function(e)
{
  if(this.readyState == 4 && this.status == 200)
  {
  _id_Vopros = this.response;
  document.location.href='#openVopros';
  }
  else
  {
  alert("Err!");
  }
};
xhr.send(formData);
}
```



```
/* ********************************************************** */
// Удаление документа из коллекции в базе данных
function RemoveDoc(sNameDB, sNameCollection, s_id)
{
var formData = new FormData();
formData.append("namedb", sNameDB);
formData.append("namecollection", sNameCollection);
formData.append("_id", s_id);

var xhr = new XMLHttpRequest();
xhr.open("POST", "remove.php");
xhr.onload = function(e)
{
  if(this.status == 200)
  {
  ;
  }
  else
  {
  alert("Err!");
  }
};
xhr.send(formData);
}

/* ********************************************************** */
// Загружает данные, возвращаемые исполняемым файлом, в div
function loadFileToDiv(sNameFile, sIDDiv, sFilter)
{
var formData = new FormData();
formData.append("filter", sFilter);

var xhr = new XMLHttpRequest();
xhr.open("POST", sNameFile);
xhr.onload = function(e)
{
  if(this.status == 200)
  {
  document.getElementById(sIDDiv).innerHTML = this.response;
  }
  else
  {
  alert("Err!");
  }
};
xhr.send(formData);
}
```

```
</script>

</head>

<body onLoad="loadOption('school', 'tema', 'tema', 'tema',
'idpredmet='+predmet.value);">

<!-- При загрузке страницы из базы данных school передаются
значения для заполнения раскрывающегося списка тем,
соответствующие тому предмету, который выбран в списке
предметов -->

<div id="openPredmet" class="modalDialog">
  <div>
  <a href="#close" title="Close" class="close">X</a>
  <h3>Предметы</h3>
  <p>
  <form id=formop>
  <span class=label1>Название предмета:</span><br>
  <input type="text" id="nazvpredm" size=20 onChange=
  "SetField('school', 'predmet', _id_Predmet, 'nazvpredm',
this.value, 1);
  loadOption('school', 'predmet', 'predmet', 'predm', '');
  loadOption('school', 'tema', 'tema', 'tema',
'idpredmet='+predmet.value);"
  ><br><br>
  <span class=label1 id=tspredm>Требование
стандарта:</span><br>
  <textarea rows="5" cols="50" id="trebstandpredm" onChange=
  "SetField('school', 'predmet', _id_Predmet,
'trebstandpredm', this.value, 1);
  loadOption('school', 'predmet', 'predmet', 'predm', '');
  loadOption('school', 'tema', 'tema', 'tema',
'idpredmet='+predmet.value);"></textarea>
  </form>
  <div id=div1>
  </div>

  </div>
</div>

<div id="openTema" class="modalDialog">
  <div>
  <a href="#close" title="Close" class="close">X</a>
  <h3>Тема</h3>
  <p>
  <form id=formot>
  <span class=label1>Название темы:</span><br>
  <input type="text" id="nazvtema" size=20 onChange=
```



```
  "SetField('school', 'tema', _id_Tema, 'nazvtema',
this.value, 1);
  loadOption('school', 'tema', 'tema', 'tema',
'idpredmet='+predmet.value); "><br><br>
  <span class=label1>Часы:</span><br>
  <input type="text" id="chasitema" size=7 onChange=
  "SetField('school', 'tema', _id_Tema, 'chasitema',
this.value, 1);
  loadOption('school', 'tema', 'tema', 'tema',
'idpredmet='+predmet.value);"><br><br>
  </form>
  </div>
</div>

<div id="openVopros" class="modalDialog">
  <div>
  <a href="#close" title="Close" class="close">X</a>
  <h3>Вопрос</h3>
  <p>
  <form id=formov>
  <span class=label1>Номер вопроса:</span><br>
  <input type="text" id="nomvopros" size=20 onChange=
  "SetField('school', 'vopros', _id_Vopros, 'nomvopros',
this.value, 1);
  loadFileToDiv('vopros.php', 'divvopros',
'idtema='+tema.value);"><br><br>
  <span class=label1>Вопрос:</span><br>
  <textarea rows="5" cols="50" id="nazvvopros" onChange=
  "SetField('school', 'vopros', _id_Vopros, 'nazvvopros',
this.value, 1);
  loadFileToDiv('vopros.php', 'divvopros',
'idtema='+tema.value);"></textarea><br><br>
  </form>
  </div>
</div>

<table border>
<tr>
<td>
<form action="">
<select id="predmet" onChange="
loadOption('school', 'tema', 'tema', 'tema',
'idpredmet='+predmet.value);"
>
</select> 
<button onClick="_id_Predmet=-1;
document.getElementById('formop').reset();
AddPredm();
return false;">+</button>
```

```
<button onClick="_id_Predmet=predmet.value;
if(_id_Predmet.length > 20)
{
GetField('school', 'predmet', _id_Predmet, 'nazvpredm',
'nazvpredm');
GetField('school', 'predmet', _id_Predmet, 'trebstandpredm',
'trebstandpredm');
document.location.href='#openPredmet';
}">&frasl;</button>
<button onClick="_id_Predmet=predmet.value;
if(_id_Predmet.length > 20)
{
RemoveDoc('school', 'predmet', _id_Predmet);
loadOption('school', 'predmet', 'predmet', 'predm', '');
return false;
}">&minus;</button>
<button onClick="loadFileToDiv('predm.php', 'divforloadfile',
''); return false;">&equiv;</button>
<br>
<select id="tema" onChange="
loadFileToDiv('vopros.php', 'divvopros',
'idtema='+tema.value);">
</select> 
<button onClick="
if(predmet.value.length > 20)
{
_id_Tema=-1; document.getElementById('formot').reset();
_id_Predmet=predmet.value;
AddTema();
return false;
}
else { alert('Выберите предмет!'); }">+</button>
<button onClick="_id_Tema=tema.value;
if(_id_Tema.length > 20)
{
GetField('school', 'tema', _id_Tema, 'nazvtema', 'nazvtema');
GetField('school', 'tema', _id_Tema, 'chasitema',
'chasitema');
document.location.href='#openTema';
}">&frasl;</button>
<button onClick="_id_Tema=tema.value;
if(_id_Tema.length > 20)
{
RemoveDoc('school', 'tema', _id_Tema);
loadOption('school', 'tema', 'tema', 'tema',
'idpredmet='+predmet.value); return false;
}">&minus;</button>
<button onClick="loadFileToDiv('tema.php', 'divforloadfile',
'idpredmet='+predmet.value); return false;">&equiv;</button>
```



```
</form>
</td>
<td>
<div id="divforloadfile"></div>
</td>
</tr>
</table>
<br> <br>
<div id="divvopros"></div>
<script>
loadOption('school', 'predmet', 'predmet', 'predm','');
</script>
</body>
</html>
```

Функции добавления предметов, тем и вопросов находятся в файлах **dobavpredm.php**, **dobavtema.php** и **dobavvopros.php** соответственно. Эти функции вызываются в соответствующем месте файла **index.htm** для обмена информацией с сервером без перезагрузки страницы через экземпляр встроенного объекта **XMLHttpRequest**.

Код файла **dobavpredm.php**:

```
<?php
$sNameDB = "school";
$sNameCollection = "predmet";
try {
// установить соединение с сервером MongoDB
 $conn = new Mongo('localhost');
// подключиться к базе данных
 $db = $conn->$sNameDB;
// выбрать коллекцию
 $collection = $db->$sNameCollection;
 //
 $item['nazvpredm'] = "";
 $item['trebstandpredm'] = "";
 $collection->insert($item);
 echo $item['_id'];
// закрыть соединение с сервером
$conn->close();
}
// обработка исключений
catch (MongoConnectionException $e) {
die('Error connecting to MongoDB server');
} catch (MongoException $e) {
die('Error: ' . $e->getMessage());
}
?>
```

Код файла **dobavtema.php**:

```
<?php
$idpredmet = $_POST['idpredmet'];
$sNameDB = "school";
$sNameCollection = "tema";
try {
// установить соединение с сервером MongoDB
 $conn = new Mongo('localhost');
// подключиться к базе данных
 $db = $conn->$sNameDB;
// выбрать коллекцию
 $collection = $db->$sNameCollection;
 //
 $item['idpredmet'] = $idpredmet;
 $item['nazvtema'] = "";
 $item['chasitema'] = "";
  $collection->insert($item);
 echo $item['_id'];
// закрыть соединение с сервером
$conn->close();
}
// обработка исключений
catch (MongoConnectionException $e) {
die('Error connecting to MongoDB server');
} catch (MongoException $e) {
die('Error: ' . $e->getMessage());
}
?>
```

Код файла **dobavvopros.php**:

```
<?php
$idtema = $_POST['idtema'];
$sNameDB = "school";
$sNameCollection = "vopros";
try {
// установить соединение с сервером MongoDB
$conn = new Mongo('localhost');
// подключиться к базе данных
 $db = $conn->$sNameDB;
// выбрать коллекцию
 $collection = $db->$sNameCollection;
 //
 $item['idtema'] = $idtema;
 $item['nomvopros'] = "";
 $item['nazvvopros'] = "";
  $collection->insert($item);
 echo $item['_id'];
```



```
// закрыть соединение с сервером
$conn->close();
}
// обработка исключений
catch (MongoConnectionException $e) {
die('Error connecting to MongoDB server');
} catch (MongoException $e) {
die('Error: ' . $e->getMessage());
}
?>
```

Файл **lo.php** предназначен для формирования раскрывающегося списка:

```
<?php

$sNameDB = $_POST['namedb'];
$sNameCollection = $_POST['namecollection'];
$sSufNazv = $_POST['sufnazv'];
$sForFilter = $_POST['filter'];

try {
// установить соединение с сервером MongoDB
$conn = new Mongo('localhost');
// подключиться к базе данных
 $db = $conn->$sNameDB;
// выбрать коллекцию
 $collection = $db->$sNameCollection;
// execute query
// retrieve all documents
parse_str($sForFilter, $filter);
$cursor = $collection->find($filter);
// перебор и печать всех возвращенных документов
foreach ($cursor as $obj) {
echo '<option
value="'.$obj['_id'].'">'.$obj["nazv$sSufNazv"].'</option>'; }
// закрыть соединение с сервером
$conn->close();
}
// обработка исключений
catch (MongoConnectionException $e) {
die('Error connecting to MongoDB server');
} catch (MongoException $e) {
die('Error: ' . $e->getMessage());
}
?>
```

Файлы **predm.php**, **tema.php** и **vopros.php** выводят содержимое каждого документа из соответствующей коллекции «Предмет», «Тема» и «Вопрос».

Код файла **predm.php**:

```
<?php
$sNameDB = "school";
$sNameCollection = "predmet";
try {
 // установить соединение с сервером MongoDB
 $conn = new Mongo('localhost');
// подключиться к базе данных
 $db = $conn->$sNameDB;
// выбрать коллекцию
 $collection = $db->$sNameCollection;
// выполнение запроса и получение с сервера документов
$cursor = $collection->find();
// перебор и печать всех возвращенных документов
echo $cursor->count() . ' всего предметов. <br/>';
foreach ($cursor as $obj)
{
  foreach ($obj as $k=>$v)
  {
  echo "$k: $v <BR>";
  }
}
// закрыть соединение с сервером
$conn->close();
}
// обработка исключений
catch (MongoConnectionException $e) {
die('Error connecting to MongoDB server');
} catch (MongoException $e) {
die('Error: ' . $e->getMessage());
}
?>
```

Код файла **tema.php**:

```
<?php
$sNameDB = "school";
$sNameCollection = "tema";
$sForFilter = $_POST['filter'];
echo "sForFilter=$sForFilter<BR>";

try {
// установить соединение с сервером MongoDB
 $conn = new Mongo('localhost');
// подключиться к базе данных
 $db = $conn->$sNameDB;
// выбрать коллекцию
 $collection = $db->$sNameCollection;
// выполнение запроса и получение с сервера документов
parse_str($sForFilter, $filter);
$cursor = $collection->find($filter);
// перебор и печать всех возвращенных документов
echo $cursor->count() . ' всего предметов. <br/>';
foreach ($cursor as $obj)
{
  foreach ($obj as $k=>$v)
  {
  echo "$k: $v <BR>";
  }
}

// закрыть соединение с сервером
$conn->close();
}
// обработка исключений
catch (MongoConnectionException $e) {
die('Error connecting to MongoDB server');
} catch (MongoException $e) {
die('Error: ' . $e->getMessage());
}
?>
```


Код файла **vopros.php**:

```
<?php
$sNameDB = "school";
$sNameCollection = "vopros";
$sForFilter = $_POST['filter'];
?>

<button onClick="
if(tema.value.length > 20)
{
_id_Vopros=-1; document.getElementById('formov').reset();
_id_Tema=tema.value;
AddVopros();
return false;
}
else { alert('Выберите тему!');
}">+</button>   

<?php
try {
 // установить соединение с сервером MongoDB
 $conn = new Mongo('localhost');
```

```
// подключиться к базе данных
 $db = $conn->$sNameDB;
// выбрать коллекцию
 $collection = $db->$sNameCollection;
// выполнение запроса и получение с сервера документов
parse_str($sForFilter, $filter);
$cursor = $collection->find($filter)->sort(array('nomvopros'
=> 1));
// перебор и печать всех возвращенных документов
echo $cursor->count() . ' всего вопросов. <br/><br/>';
foreach ($cursor as $obj)
{
$cur_id = $obj['_id'];

echo $obj['nomvopros']."." ";
echo $obj['nazvvopros']."  ";
?>

<button onClick="
_id_Vopros='<?php echo $cur_id; ?>';
GetField('school', 'vopros', _id_Vopros, 'nomvopros',
'nomvopros');
GetField('school', 'vopros', _id_Vopros, 'nazvvopros',
'nazvvopros');
document.location.href='#openVopros';
">&frasl;</button>

<button onClick="
_id_Vopros='<?php echo $cur_id; ?>';
RemoveDoc('school', 'vopros', _id_Vopros);
loadFileToDiv('vopros.php', 'divvopros',
'idtema='+tema.value);
">&minus;</button>
<br>
<?php
}
// закрыть соединение с сервером
$conn->close();
}
// обработка исключений
catch (MongoConnectionException $e) {
die('Error connecting to MongoDB server');
} catch (MongoException $e) {
die('Error: ' . $e->getMessage());
}
?>
```


Файл **remove.php** предназначен для удаления документов:

```
<?php
$sNameDB = $_POST['namedb'];
$sNameCollection = $_POST['namecollection'];
$s_id = $_POST['_id'];
echo "sNameDB=$sNameDB <BR>";
echo "sNameCollection=$sNameCollection <BR>";
echo "s_id=$s_id <BR>";

try {
 // установить соединение с сервером MongoDB
 $conn = new Mongo('localhost');
// подключиться к базе данных
 $db = $conn->$sNameDB;
// выбрать коллекцию
 $collection = $db->$sNameCollection;
 $filter = array('_id'=> new MongoId($s_id));
 $collection->remove($filter);
 echo 'Remove document with ID: ' . $s_id;
 // закрыть соединение с сервером
 $conn->close();
}
// обработка исключений
catch (MongoConnectionException $e) {
 die('Error connecting to MongoDB server');
} catch (MongoException $e) {
 die('Error: ' . $e->getMessage());
}
```

Файл **getfield.php** предназначен для получения значения поля документа из соответствующей коллекции:

```
<?php
$sNameDB = $_POST['namedb'];
$sNameCollection = $_POST['namecollection'];
$s_id = $_POST['_id'];
$sNameField = $_POST['namefield'];
try {
 // установить соединение с сервером MongoDB
 $conn = new Mongo('localhost');
// подключиться к базе данных
 $db = $conn->$sNameDB;
// выбрать коллекцию
 $collection = $db->$sNameCollection;
```

```
// вставить новый документ
$item["$sNameField"] = $sValueField;
$filter = array('_id'=> new MongoId($s_id));
$cursor = $collection->find($filter);
foreach ($cursor as $obj)
{
echo $obj["$sNameField"];
break;
}
 // закрыть соединение с сервером
 $conn->close();
}
// обработка исключений
catch (MongoConnectionException $e) {
 die('Error connecting to MongoDB server');
} catch (MongoException $e) {
 die('Error: ' . $e->getMessage());
}
?>
```

Файл **setfield.php** предназначен для замены в соответствующей коллекции соответствующего поля:

```
<?php
$sNameDB = $_POST['namedb'];
$sNameCollection = $_POST['namecollection'];
$s_id = $_POST['_id'];
$sNameField = $_POST['namefield'];
$sValueField = $_POST['valuefield'];
$nSignText = $_POST['signtext'];

echo "sNameDB=$sNameDB <BR>";
echo "sNameCollection=$sNameCollection <BR>";
echo "s_id=$s_id <BR>";
echo "sNameField=$sNameField <BR>";
echo "sValueField=$sValueField <BR>";
echo "nSignText=$nSignText <BR>";

try {
 // установить соединение с сервером MongoDB
 $conn = new Mongo('localhost');
// подключиться к базе данных
 $db = $conn->$sNameDB;
// выбрать коллекцию
 $collection = $db->$sNameCollection;
// вставить новый документ
 $item["$sNameField"] = $sValueField;
 print_r($item);
 $filter = array('_id'=> new MongoId($s_id));
```



```
 $update = array('$set'=>$item);
 $collection->update($filter, $update);
 echo 'Update document with ID: ' . $s_id;
 // закрыть соединение с сервером
 $conn->close();
}
// обработка исключений
catch (MongoConnectionException $e) {
 die('Error connecting to MongoDB server');
} catch (MongoException $e) {
 die('Error: ' . $e->getMessage());
}

?>
```

Контрольные вопросы

1. Каким образом при помощи HTML5 и CSS3 создается модальное окно? Для чего оно предназначено в файле **index.htm**?
2. Каким образом позиционируется модальное окно? Как сделать, чтобы его положение оставалось неподвижным при прокрутке?
3. Для каких целей применяется объект **XMLHttpRequest**? Каковы принципы его использования?
4. Какой метод передачи данных используется **GET** или **POST**?
5. При помощи какой команды происходит установление соединения с сервером MongoDB? С базой данных?
6. Для чего служат функции **function GetField** и **function SetField**?
7. При помощи какой команды закрывается соединение с сервером?
8. Какие функции служат для обработки исключений?

Заключение

В области технологий обработки данных ситуация изменяется достаточно быстро. Пройдет, возможно, несколько месяцев или лет, и приоритеты могут существенно измениться, что связано как с деятельностью разработчиков СУБД, так и с политикой лицензирования крупных компаний — владельцев ПО. Не случайно опытные пользователи знают, что достаточно часто (раз в несколько лет) приходится переносить данные (либо целые информационные системы) с одной платформы на другую. При этом достаточно мощным и динамически развивающимся является направление, связанное со свободным программным обеспечением, впитывающим новые тенденции и активно отвечающим на современные вызовы. Так, свободное программное обеспечение охватывает две большие области, связанные с разработкой информационных систем с использованием СУБД, — на основе SQL и NoSQL-типа. Таким образом, изучение представленных в данном учебном пособии основ работы с представителями обоих типов СУБД является актуальным.

Коллектив авторов надеется, что предлагаемое учебное пособие принесет существенную пользу учащимся при изучении современных систем обработки информации и построения баз данных.

Пособие не свободно от недостатков, и все замечания и предложения авторы примут с искренней признательностью.

Литература

1. *Мартишин С.А., Симонов В.Л., Храпченко М.В.* Проектирование и реализация баз данных в СУБД MySQL с использованием MySQL Workbench. Методы и средства проектирования информационных систем и технологий. Инструментальные средства информационных систем: учеб. пособие. М.: ИД «ФОРУМ»; ИНФРА-М, 2012.

2. *Майоров А.А., Соловьев И.В.* Проектирование информационных систем: учеб. пособие для вузо. Академический Проект. Сер.: Gaudeamus. 2009.

3. *Бенкер К.* MongoDB в действии / пер. с англ. М.: ДМК Пресс, 2014.

4. *Фаулер М., Садаладж П.Дж.* NoSQL: новая методология разработки нереляционных баз данных — NoSQL Distilled. М.: Вильямс, 2015.

5. BSON [Электронный ресурс]. URL: http://bsonspec.org/ (Дата обращения: 25.01.2016.)

6. MongoDB [Электронный ресурс]. URL: https://www.mongodb.org/ (Дата обращения: 25.01.2016.)

7. MariaDB [Электронный ресурс]. URL: https://mariadb.org/ (Дата обращения: 25.01.2016.)

8. *Payne K.* Creating a modal window with HTML5 & CSS3 / Keenan Payne // Webdesigner Depot [Электронный ресурс]. 2012. 2 окт. URL: http://www.webdesignerdepot.com/2012/10/creating-a-modal-window-with-html5-and-css3 (Дата обращения: 25.01.2016.)

9. World Wide Web Consortium W3C [Электронный ресурс]. URL: http://www.w3.org/ (Дата обращения: 25.01.2016.)

10. Welcome to IDEF.com [Электронный ресурс]. URL: http://www.idef.com/Home.htm (Дата обращения: 25.01.2016.)

11. XMLHttpRequest Living Standard [Электронный ресурс]. Last Updated 20 January 2016. URL: https://xhr.spec.whatwg.org/ (Дата обращения: 25.01.2016.)

12. Robomongo [Электронный ресурс]. URL: http://robomongo.org (Дата обращения: 25.01.2016.)

13. DBeaver [Электронный ресурс]. URL: http://dbeaver.jkiss.org/ (Дата обращения: 25.01.2016).

Приложения

Приложение А

ЦЕЛОСТНОСТЬ ДАННЫХ

Выполнение операторов модификации данных в таблицах базы данных INSERT, DELETE и UPDATE может привести к нарушению **целостности данных** и их корректности, т. е. к потере их достоверности и непротиворечивости.

Чтобы избежать такого рода нарушений, в реляционной модели устанавливаются некоторые правила целостности, которые, по сути, являются ограничениями для всех допустимых состояний базы данных и гарантируют корректность данных. **Ограничения целостности** позволяют свести к минимуму ошибки, возникающие при обновлении и обработке данных.

Рассмотрим основные типы ограничений целостности данных:

- обязательные данные;
- ограничения для доменов полей;
- целостность сущностей;
- ссылочная целостность.

Обязательные данные

Некоторые поля всегда должны содержать одно из допустимых значений, эти поля не могут иметь пустого значения.

Ограничения для доменов полей

Каждое поле имеет свой домен, представляющий собой набор его допустимых значений.

Целостность сущностей

Это ограничение целостности касается первичных ключей базовых таблиц. По определению, **первичный ключ** — минимальный идентификатор (одно или несколько полей), который используется для уникальной идентификации записей в таблице. Таким образом, никакое подмножество первичного ключа не может быть достаточным для уникальной идентификации записей.

Целостность сущностей определяет, что в базовой таблице ни одно поле первичного ключа не может содержать неопределенных значений NULL.

Это ограничение представляется абсолютно естественным, поскольку в том случае, когда значение NULL может появиться в любой части первичного ключа, это будет означать, что не все его поля необходимы для уникальной идентификации записей, что противоречит определению первичного ключа.

Ссылочная целостность

Данное ограничение целостности касается внешних ключей. **Внешний ключ** — это поле (или множество полей) одной таблицы, являющееся ключом другой (или той же самой) таблицы. Внешние ключи используются для установления логических связей между таблицами. Связь устанавливается путем присвоения значений внешнего ключа одной таблицы значениям ключа другой.

Между двумя или более таблицами базы данных, как правило, существуют отношения подчиненности, которые определяют, что для каждой записи главной таблицы (называемой еще **родительской**) может существовать одна или несколько записей в подчиненной таблице (называемой также **дочерней**).

Существует три вида связи между таблицами базы данных:

- один-ко-многим;
- один-к-одному;
- многие-ко-многим.

Отношение один-ко-многим имеет место, когда одной записи родительской таблицы может соответствовать несколько записей дочерней. Связь один-ко-многим наиболее распространена для реляционных баз данных.

Отношение один-к-одному имеет место, когда одной записи в родительской таблице соответствует одна запись в дочерней. Это отношение встречается намного реже, чем отношение один-ко-многим. Использование связи один-к-одному приводит к тому, что для чтения связанной информации в нескольких таблицах приходится производить несколько операций чтения вместо одной, когда данные хранятся в одной таблице.

Отношение многие-ко-многим имеет место в следующих случаях:

- одной записи в родительской таблице соответствует более одной записи в дочерней таблице;
- одной записи в дочерней таблице соответствует более одной записи в родительской таблице.

Считается, что всякая связь многие-ко-многим может быть заменена на связь один-ко-многим (одну или несколько). Чаще всего связь между таблицами устанавливается по первичному ключу, т. е. значениям внеш-

него ключа одной таблицы присваиваются значения первичного ключа другой. Поля внешнего ключа не обязаны иметь те же имена, что и имена ключей, которым они соответствуют. Внешний ключ может ссылаться на свою собственную таблицу — в таком случае внешний ключ называется рекурсивным.

В общем случае, если участие дочерней таблицы в связи является обязательным, то рекомендуется запрещать применение пустых значений в соответствующем внешнем ключе. В то же время, если имеет место частичное участие дочерней таблицы в связи, то помещение пустых значений в поле внешнего ключа должно быть разрешено.

Также необходимо решить, каким образом будет организована поддержка ссылочной целостности при выполнении операций модификации данных в базе. Здесь могут возникнуть следующие случаи.

1. Вставка новой строки осуществляется в дочернюю таблицу. Для обеспечения ссылочной целостности необходимо убедиться, что значение внешнего ключа новой строки дочерней таблицы равно некоторому конкретному значению, присутствующему в поле первичного ключа одной из строк родительской таблицы.

2. Обновление внешнего ключа в строке дочерней таблицы. Этот случай подобен описанной выше вставке новой строки.

3. Удаление строки из дочерней таблицы. Никаких нарушений ссылочной целостности не происходит.

4. Вставка строки в родительскую таблицу. Такая вставка не вызывает нарушения ссылочной целостности. Добавленная строка просто становится родительским объектом, не имеющим дочерних объектов.

5. Удаление строки из родительской таблицы. Ссылочная целостность окажется нарушенной, если в дочерней таблице будут существовать строки, ссылающиеся на удаленную строку родительской таблицы. В этом случае может использоваться одна из следующих стратегий:

- **NO ACTION**. Удаление строки из родительской таблицы запрещается, если в дочерней таблице существует хотя бы одна ссылающаяся на нее строка;
- **CASCADE**. При удалении строки из родительской таблицы автоматически удаляются все ссылающиеся на нее строки дочерней таблицы. Если какая-либо из удаляемых строк дочерней таблицы выступает в качестве родительской стороны в какой-либо другой связи, то и здесь удаляются все ссылающиеся на нее строки и т. д. То есть удаление строки родительской таблицы автоматически распространяется на любые дочерние таблицы;
- **SET NULL**. При удалении строки из родительской таблицы во всех ссылающихся на нее строках дочернего отношения в поле внешнего ключа, соответствующего первичному ключу удаленной строки, записывается значение NULL. Эта стратегия может использоваться, только когда в поле внешнего ключа дочерней таблицы разрешается помещать пустые значения;
- **SET DEFAULT**. При удалении строки из родительской таблицы в поле внешнего ключа всех ссылающихся на нее строк дочерней таблицы автоматически помещается значение, указанное для этого поля как значение по умолчанию. Эта стратегия применима лишь в тех случаях, когда полю внешнего ключа дочерней таблицы назначено некоторое значение, принимаемое по умолчанию;
- **NO CHECK**. При удалении строки из родительской таблицы никаких действий по сохранению ссылочной целостности данных не предпринимается.

6. Обновление первичного ключа в строке родительской таблицы. Если значение первичного ключа некоторой строки родительской таблицы будет обновлено, нарушение ссылочной целостности произойдет в том случае, если в дочерней таблице существуют строки, ссылающиеся на исходное значение первичного ключа. Для сохранения ссылочной целостности может применяться любая из описанных выше стратегий.

Различные СУБД по-разному реализуют поддержку целостности данных. При выборе конкретных стратегий следует внимательно изучить механизм реализации целостности данных конкретной СУБД. С развитием архитектуры клиент-сервер появилась возможность переноса максимально возможного числа правил целостности данных на сервер. Это имеет как определенные преимущества, так и некоторые недостатки, но их обсуждение выходит за рамки данного изложения.

Приложение В

СЕМЕЙСТВО СТАНДАРТОВ IDEF

В настоящий момент к семейству IDEF можно отнести следующие стандарты:

IDEF0 — Function Modeling. Методология функционального моделирования. С помощью наглядного графического языка IDEF0 изучаемая система предстает перед разработчиками и аналитиками в виде набора взаимосвязанных функций (функциональных блоков в терминах IDEF0). Как правило, моделирование средствами IDEF0 является первым этапом

изучения любой системы. Методологию IDEF0 можно считать следующим этапом развития хорошо известного графического языка описания функциональных систем SADT (Structured Analysis and Design Teqnique);

IDEF1 — Information Modeling. Методология моделирования информационных потоков внутри системы, позволяющая отображать и анализировать их структуру и взаимосвязи;

IDEF1X (IDEF1 Extended) — Data Modeling. Методология построения реляционных структур (баз данных), относится к типу методологий «Сущность—связь» (ER—EntityRelationship) и, как правило, используется для моделирования реляционных баз данных, имеющих отношение к рассматриваемой системе;

IDEF2 — Simulation Model Design. Методология динамического моделирования развития систем. В связи с весьма серьезными сложностями анализа динамических систем от этого стандарта практически отказались, и его развитие приостановилось на самом начальном этапе. В настоящее время присутствуют алгоритмы и их компьютерные реализации, позволяющие превращать набор статических диаграмм IDEF0 в динамические модели, построенные на базе «раскрашенных сетей Петри» (CPN — Color Petri Nets);

IDEF3 — Process Description Capture. Документирование технологических процессов, IDEF3 — методология документирования процессов, происходящих в системе (например, на предприятии), описываются сценарий и последовательность операций для каждого процесса. IDEF3 имеет прямую взаимосвязь с методологией IDEF0 — каждая функция (функциональный блок) может быть представлена в виде отдельного процесса средствами IDEF3;

IDEF4 — Object Oriented Design. Методология построения объектно-ориентированных систем, позволяет отображать структуру объектов и заложенные принципы их взаимодействия, тем самым позволяя анализировать и оптимизировать сложные объектно-ориентированные системы;

IDEF5 — Ontology Description Capture. Стандарт онтологического исследования сложных систем. С помощью методологии IDEF5 онтология системы может быть описана при помощи определенного словаря терминов и правил, на основании которых могут быть сформированы достоверные утверждения о состоянии рассматриваемой системы в некоторый момент времени. На основе этих утверждений формируются выводы о дальнейшем развитии системы и производится ее оптимизация;

IDEF6 — Design Rationale Capture. Обоснование проектных действий. Назначение IDEF6 состоит в облегчении получения «знаний о способе» моделирования, их представления и использования при разработке систем управления предприятиями. Под «знаниями о способе» понимаются причины, обстоятельства, скрытые мотивы, которые обусловливают

выбранные методы моделирования. Проще говоря, «знания о способе» интерпретируются как ответ на вопрос: «Почему модель получилась такой, какой получилась?» Большинство методов моделирования фокусируются на собственно получаемых моделях, а не на процессе их создания. Метод IDEF6 акцентирует внимание именно на процессе создания модели;

IDEF7 — Information System Auditing. Аудит информационных систем. Этот метод определен как востребованный, однако так и не был полностью разработан;

IDEF8 — User Interface Modeling. Метод разработки интерфейсов взаимодействия оператора и системы (пользовательских интерфейсов). Современные среды разработки пользовательских интерфейсов в большей степени создают внешний вид интерфейса. IDFE8 фокусирует внимание разработчиков интерфейса на программировании желаемого взаимного поведения интерфейса и пользователя на трех уровнях: выполняемой операции (что это за операция); сценарии взаимодействия, определяемом специфической ролью пользователя (по какому сценарию она должна выполняться тем или иным пользователем); и, наконец, на деталях интерфейса (какие элементы управления предлагает интерфейс для выполнения операции);

IDEF9 — ScenarioDriven IS Design (Business Constraint Discovery method). Метод исследования бизнес-ограничений был разработан для облегчения обнаружения и анализа ограничений, в условиях которых действует предприятие. Обычно при построении моделей описанию ограничений, оказывающих влияние на протекание процессов на предприятии, уделяется недостаточное внимание. Знания об основных ограничениях и характере их влияния, закладываемые в модели, в лучшем случае остаются неполными, несогласованными, распределенными нерационально, но часто их вовсе нет. Это не обязательно приводит к тому, что построенные модели нежизнеспособны, просто их реализация столкнется с непредвиденными трудностями, в результате чего их потенциал будет не реализован. Тем не менее в случаях, когда речь идет именно о совершенствовании структур или адаптации к предсказываемым изменениям, знания о существующих ограничениях имеют критическое значение;

IDEF10 — Implementation Architecture Modeling. Моделирование архитектуры выполнения. Этот метод определен как востребованный, однако так и не был полностью разработан;

IDEF11 — Information Artifact Modeling. Этот метод определен как востребованный, однако так и не был полностью разработан;

IDEF12 — Organization Modeling. Организационное моделирование. Этот метод определен как востребованный, однако так и не был полностью разработан;

IDEF13 — Three Schema Mapping Design. Трехсхемное проектирование преобразования данных. Этот метод определен как востребованный, однако так и не был полностью разработан;

IDEF14 — Network Design. Метод проектирования компьютерных сетей, основанный на анализе требований, специфических сетевых компонентов, существующих конфигураций сетей. Также он обеспечивает поддержку решений, связанных с рациональным управлением материальными ресурсами, что позволяет достичь существенной экономии.

Приложение С

ОПИСАНИЕ СПЕЦИФИКАЦИИ BSON

Основные типы

byte	1 byte (8-bits)
int32	4 bytes (32-bit signed integer)
int64	8 bytes (64-bit signed integer)
double	8 bytes (64-bit IEEE 754 floating point)

Определение грамматики BSON

document	::=	int32 e_list "\x00"	BSON Document
e_list	::=	element e_list	Sequence of elements
	|	""	
element	::=	"\x01" e_name double	Floating point
	|	"\x02" e_name string	UTF-8 string
	|	"\x03" e_name document	Embedded document
	|	"\x04" e_name document	Array
	|	"\x05" e_name binary	Binary data
	|	"\x06" e_name	Undefined — *Deprecated*
	|	"\x07" e_name (byte*12)	ObjectId
	|	"\x08" e_name "\x00"	Boolean «false»

Окончание табл.

	|	"\x08" e_name "\x01"	Boolean «true»
	|	"\x09" e_name int64	UTC datetime
	|	"\x0A" e_name	Null value
	|	"\x0B" e_name cstring cstring	Regular expression
	|	"\x0C" e_name string (byte*12)	DBPointer — *Deprecated*
	|	"\x0D" e_name string	JavaScript code
	|	"\x0E" e_name string	Symbol — *Deprecated*
	|	"\x0F" e_name code_w_s	JavaScript code w/scope
	|	"\x10" e_name int32	32-bit integer
	|	"\x11" e_name int64	Timestamp
	|	"\x12" e_name int64	64-bit integer
	|	"\xFF" e_name	Min key
	|	"\x7F" e_name	Max key
e_name	::=	cstring	Key name
string	::=	int32 (byte*) "\x00"	String
cstring	::=	(byte*) "\x00"	CString
binary	::=	int32 subtype (byte*)	Binary
subtype	::=	"\x00"	Binary / Generic
	|	"\x01"	Function
	|	"\x02"	Binary (Old)
	|	"\x03"	UUID (Old)
	|	"\x04"	UUID
	|	"\x05"	MD5
	|	"\x80"	User defined
code_w_s	::=	int32 string document	Code w/scope

BSON: соответствие тип данных — число

Тип данных	Число	Псевдоним
Double	1	«double»
String	2	«string»
Object	3	«object»
Array	4	«array»

Окончание табл.

Тип данных	Число	Псевдоним
Binary data	5	«binData»
Undefined	6	«undefined»
Object id	7	«objectId»
Boolean	8	«bool»
Date	9	«date»
Null	10	«null»
Regular Expression	11	«regex»
DBPointer	12	«dbPointer»
JavaScript	13	«javascript»
Symbol	14	«symbol»
JavaScript (with scope)	15	«javascriptWithScope»
32-bit integer	16	«int»
Timestamp	17	«timestamp»
64-bit integer	18	«long»
Min key	–1	«minKey»
Max key	127	«maxKey»

Приложение D

СЕЛЕКТОРЫ ЗАПРОСА (QUERY SELECTORS)

Сравнение

Имя	Описание
$gt	Соответствует значениям, которые больше указанных в запросе
$gte	Соответствует значениям, которые больше указанных в запросе или равны им
$in	Соответствует значениям, которые существуют среди множества значений, указанных в запросе

Окончание табл.

Имя	Описание
$lt	Соответствует значениям, которые меньше указанных в запросе
$lte	Соответствует значениям, которые меньше указанных в запросе или равны им
$ne	Соответствует значениям, которые не равны указанным в запросе
$nin	Соответствует значениям, которые не существуют среди множества значений, указанных в запросе

Логические

Имя	Описание
$or	Объединяет условия запроса логическим оператором OR и возвращает все документы, соответствующие хотя бы одному условию запроса
$and	Объединяет условия запроса логическим оператором AND и возвращает все документы, соответствующие всем условиям запроса
$not	Инвертирует условие запроса и возвращает все документы, соответствующие инвертированному условию запроса
$nor	Объединяет условия запроса логическим оператором NOR и возвращает все документы, соответствующие инверсии всех условий запроса

Элемент

Имя	Описание
$exists	Соответствует документам, в которых содержится заданное поле
$type	Выбирает документы, если поле является полем заданного типа

Оценка

Имя	Описание
$mod	Выполняется операция остатка от деления значений полей и выбираются документы с заданным результатом

Содержание

По вопросам приобретения книг обращайтесь:
Отдел продаж «ИНФРА-М» (оптовая продажа):
127282, Москва, ул. Полярная, д. 31В, стр.1
Тел. (495) 280-15-96; факс (495) 280-36-29
E-mail: books@infra-m.ru

•

Отдел «Книга–почтой»:
тел. (495) 280-15-96 (доб. 246)

ФЗ № 436-ФЗ	Издание не подлежит маркировке в соответствии с п. 1 ч. 4 ст. 11

Учебное издание

Мартишин Сергей Анатольевич,
Симонов Владимир Львович,
Храпченко Марина Валерьевна

Базы данных.

Практическое применение СУБД SQL- и NoSQL-типа для проектирования информационных систем

УЧЕБНОЕ ПОСОБИЕ

ООО «Издательский дом ФОРУМ»
127247, Москва, ул. Софьи Ковалевской, д. 1, стр. 51
E-mail: forum2-book@yandex.ru
Тел.: (495) 280-15-96

ООО «Научно-издательский центр ИНФРА-М»
127282, Москва, ул. Полярная, д. 31В, стр. 1
Тел.: (495) 280-15-96, 280-33-86. Факс: (495) 280-36-29
E-mail: books@infra-m.ru http://www.infra-m.ru

Подписано в печать 04.07.2018.
Формат 60×90/16. Бумага офсетная. Гарнитура *Times*.
Печать цифровая. Усл. печ. л. 23,0.
ППТ50. Заказ № 06741
ТК 682830-988131-240516

Отпечатано в типографии ООО «Научно-издательский центр ИНФРА-М»
127282, Москва, ул. Полярная, д. 31В, стр. 1
Тел.: (495) 280-15-96, 280-33-86. Факс: (495) 280-36-29