Е. Е. Бизянов, Н. Н. Кононенко

БАЗЫ ДАННЫХ

Лабораторный практикум

Лабораторный практикум

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ДОНБАССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА СПЕЦИАЛИЗИРОВАННЫХ КОМПЬЮТЕРНЫХ СИСТЕМ

Е. Е. Бизянов, Н. Н. Кононенко

БАЗЫ ДАННЫХ

Лабораторный практикум

Рекомендовано Ученым советом
факультета автоматизации и электротехнических систем

Алчевск
ГОУ ВПО ЛНР «ДонГТУ»
2018

УДК 004.65
ББК 32.973-18.02
Б59

Рецензент
С. В. Гонтовой — к.т.н., доц. кафедры специализированных компьютерных систем ГОУ ВПО ЛНР «ДонГТУ»

Рекомендовано Ученым советом
факультета автоматизации и электротехнических систем
(Протокол № 10 от 18.06.2018)

Бизянов Е. Е.

Б59 Базы данных : лаб. практикум / Е. Е. Бизянов, Н. Н. Кононенко. — Алчевск : ГОУ ВПО ЛНР «ДонГТУ», 2018. — 186 с.

В практикуме приведены основы теории реляционных баз данных, рассмотрена система управления базами данных MySQL Server, приведены подходы к разработке настольных приложений и приложений с WEB-интерфейсом для работы с базами данных на языках программирования C# и Java.

Для студентов 3 курса направления подготовки 09.03.01 «Информатика и вычислительная техника» всех форм обучения.

УДК 004.45
ББК 32.973-18.02

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В практикуме рассмотрены основные теоретические и практические вопросы по следующим темам:

– управление базами данных, таблицами, индексами;
– запросы, триггеры, хранимые процедуры;
– программирование локальных баз данных в Visual C#.NET;
– программирование сетевых баз данных в Visual C# .NET;
– работа с базами данных в Java.

В практикуме на примерах рассматривается реализация части приводимых заданий, а так же предлагаются варианты заданий для самостоятельного выполнения студентами.

Лабораторные работы приведены в порядке возрастающей сложности.

После описания каждой работы приводится перечень контрольных вопросов для проверки знаний, умений и навыков студентов, приобретаемых при подготовке к занятиям и при выполнении работ. Эти вопросы можно использовать при собеседовании со студентами в целях контроля и активизации их работы.

1 ЛАБОРАТОРНАЯ РАБОТА № 1

Тема: сервер баз данных MySQL, управление базами данных, таблицами, индексами.

Цель работы: закрепить теоретический материал по работе с базами данных, таблицами, индексами.

1.1 Базы данных. Основные понятия и определения

База данных (БД) — поименованная структурированная совокупность взаимосвязанных данных, относящихся к определенной предметной области и находящихся под централизованным программным управлением. Программное управление осуществляется специальной программой — системой управления базами данных (СУБД), с помощью которой пользователь может определять, создавать, поддерживать БД и осуществлять к ней контролируемый доступ.

Предметной областью называется часть реального мира, представляющая интерес для использования и отраженная в информационной системе.

В настоящее время наибольшее распространение получили реляционные базы данных. Американский математик Е.Ф. Кодд в 1970 году в своей статье «Реляционная модель данных для больших совместно используемых банков данных» впервые сформулировал основные понятия и ограничения реляционной модели, ограничив набор операций в ней семью основными и одной дополнительной операцией.

Основные положения реляционной модели данных:

– высокая степень независимости от данных, обеспечивает независимость прикладных программ от внутреннего представления данных (изменения способа организации данных, путей доступа и т.п.);

– решение проблем непротиворечивости и избыточности данных путем нормализации отношений;

– расширение возможностей как управление данными за счет добавления операций над множествами.

Реляционная модель базируется на математическом понятии *отношения*, представленного таблицей. Именно поэтому модель получила название реляционной (от англ. relation — отношение).

Рассмотрим пример базы данных, содержащей две таблицы (рисунок 1.1). В первой таблице (DEPT — отделение) — содержится информация о подразделениях фирмы: DEPT — код подразделения; DNAME — название подразделения, BUDGET — годовой бюджет подразделения. Во вторую таблицу заносятся данные о рабочих: EMP — код рабочего, ENAME — фамилия рабочего, DEPT — код подразделения, в котором работает работник, SALARY — годовая заработная плата рабочего.

DEPT

DEPT	DNAME	BUDGET
D1	Маркетинга	20M
D2	Развития	10M
D3	Исследований	5M

EMP

EMP	ENAME	DEPT#	SALARY
E1	Иванов	D1	20к
E2	Петров	D1	22к
E3	Семенов	D2	15к
E4	Григорьев	D2	5к

Рисунок 1.1 — Пример базы данных

Для краткости числовые данные в таблицах приведены с использованием приставок: M — миллионы, к — тысячи.

Над таблицами БД можно выполнять ряд операций. Рассмотрим некоторые из них:

1. Выборка из таблицы DEPT отделов, в которых BUDGET > 8M:

DEPT	DNAME	BUDGET
D1	Маркетинга	20M
D2	Развития	10M

2. Выборка из таблицы DEPT столбцов DEPT и BUDGET:

DEPT	BUDGET
D1	20M
D2	10M
D3	5M

3. Объединение таблиц DEPT и EMP на основе столбца DEPT:

DEPT	DNAME	BUDGET	EMP#	ENAME	SALARY
D1	Маркетинга	20M	E1	Иванов	20к
D1	Маркетинга	20M	E2	Петров	22к
D2	Развития	10M	E3	Семенов	15к
D2	Развития	10M	E4	Григорьев	5к

Как видно из приведенных примеров, результат любой операции над таблицами базы данных — новая таблица. В этом проявляется реляционное свойство замкнутости: результат операции над объектом — объект такого же рода. Таким образом, над результатом операции можно осуществить другую операцию и так далее. Иначе говоря, можно использовать вложенные выражения, в которых операнды — тоже выражения, а не объекты.

Особенностью реляционной модели является тот факт, что операции применяются ко всему множеству строк, а не к отдельно взятой строке, то есть операндами являются целые таблицы, содержащие множество строк.

Рассмотрим подробнее таблицу БД (рис. 1.2).

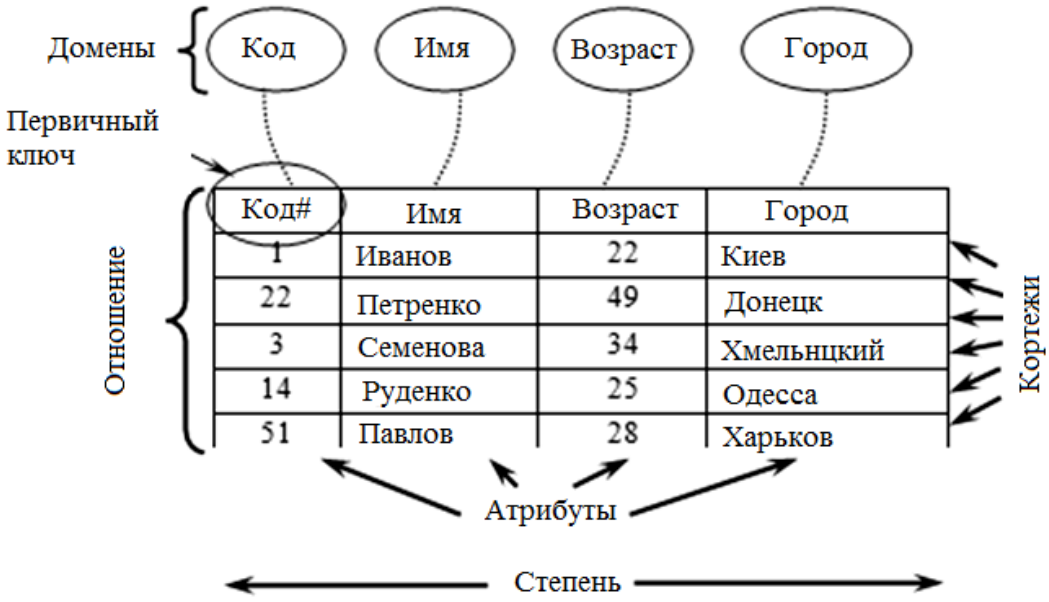

Рисунок 1.2 — Таблица реляционной базы данных

На рисунке 1.2 показаны основные составляющие отношения (таблицы БД):

Кортеж (запись) — строка таблицы;

Атрибут (поле) — столбец таблицы (поименованный столбец отношения);

Кардинальное число — количество кортежей в отношении (строк в таблице);

Степень отношения — количество столбцов (полей) в отношении.

Первый столбец (поле) отмечен знаком #, указывающим на то, что данное поле является ключевым. Различают несколько разновидностей ключей: суперключ, первичный, внешний и потенциальный ключ.

Суперключ — атрибут или множество атрибутов, однозначно идентифицирующих кортеж отношения.

Первичный ключ — уникальный идентификатор отношения (столбец, комбинация столбцов), причем такой, что для одного ключа не существует двух строк, содержащих одинаковые значения в таком столбце или комбинации столбцов.

Потенциальный ключ — это суперключ, не содержащий подмножества, который также является суперключем данного отношения.

Потенциальный ключ имеет два свойства:

– уникальность — в каждом кортеже отношения значение ключа однозначно идентифицирует этот кортеж;

– неприводимость — никакое допустимое множество ключа не имеет свойства уникальности.

Таким образом, первичный ключ — это потенциальный ключ, выбранный для уникальной идентификации доменов внутри отношения.

Внешний ключ — атрибут или множество атрибутов внутри отношения, которые соответствуют потенциальному ключу некоторого (может быть того же самого) отношения.

В приведенном ранее примере в таблице DEPT атрибут DEPT является первичным ключом, в таблице EMP — EMP является первичным ключом, DEPT — внешним.

Таким образом, ключи позволяют однозначно идентифицировать составляющие отношений и осуществлять связь между ними.

Столбец таблицы в теории реляционных БД называется доменом.

Домен — совокупность значений, из которых берутся действительные значение для определенных атрибутов определенного отношения. Так, например, домен DEPT — это множество всех допустимых значений кода подразделения, а множество значений DEPT в отношении DEPT в любой момент времени является подмножеством этого множества. Домен является наименьшей семантической единицей данных. Домены имеют свойство атомарности, то есть они неразложимые. Например, имя человека имеет внутреннюю структуру, потому что оно может быть разложено на буквы. Но, разложив его на буквы, мы потеряем значение. Таким образом, домен — это поименованное множество значений одного типа.

Домены — чрезвычайно мощный инструмент реляционной модели. Каждый атрибут реляционной БД определяется на некотором домене. Домены могут быть разными для каждого из атрибутов, но два и даже больше атрибуты могут быть определены на одном домене.

Благодаря доменам пользователь может определять содержимое и источник значений, которые могут получить атрибуты. В результате при выполнении реляционной операции системе доступно больше информации, что позволяет ей избежать семантически некорректных операций. Например, нет смысла сравнивать название улицы с номером телефона, хотя для этих атрибутов домены определены на символьных строках. С другой стороны, месячная арендная оплата объекта недвижимости и количество месяцев аренды принадлежат разным доменам (первый домен имеет денежный тип, второй, — целочисленный). Однако умножение значений из этих доменов является допустимой операцией.

Отношение R, которое определено на множестве доменов D1, D2,..., Dn (не обязательно разных), содержит 2 части: заголовок и тело (если представить отношение в терминах таблиц, то заголовок — это строка с названиями столбцов, а тело — множество строк данных).

Заголовок содержит фиксированное множество атрибутов или, точнее, пар <имя-атрибута: имя-домена>:

$$\{<A1:D1>, ,A2:D2, \ldots, ,An:Dn>\},$$

причем каждый атрибут Aj отвечает одному и только одному из доменов Dj (j=1, 2, ., n). Все имена атрибутов A1, A2, ..., An разные.

Тело содержит множество кортежей. Каждый кортеж, в свою очередь, содержит множество пар <имя-атрибута: значение-атрибута>:

{<A1:vi1>, <A2:vi2>,…,<An:vin>}, i = 1, 2, …, m,

где m - количество кортежей в этом множестве.

В каждом таком кортеже одна такая пара <имя атрибута: значение-атрибута>, то есть <Aj: vij>, для каждого атрибута Aj в заголовке. Для любой такой пары <Aj: vij> vij является значением из уникального домена Dj, который связан с атрибутом Aj. Значение m и n — соответственно кардинальное число и степень отношения R.

В качестве примера рассмотрим таблицу DEPT, которая приведена ранее, чтобы проверить, можно ли назвать ее отношением.

Во-первых, в этой таблице есть три основных домена, а именно: домен кода подразделения (DEPT#), домен имен (DNAME), домен значений бюджета (BUDGET). Представляя отношение как таблицу на листе бумаги, мы не заботимся о том, чтобы показать домены, которые лежат в основе, но должны понимать, что, по крайней мере, концептуально они всегда есть.

Дальше, в таблице обязательно есть две части: строка заголовка столбцов, а также множество строк данных. Рассмотрим сначала строку заголовков столбцов:

{DEPT#, DNAME, BUDGET}

Эта строка в действительности представляет собой набор упорядоченных пар:

{<DEPT#: DEPT#>,
 <DNAME: DNAME>,
 <BUDGETS: BUDGET>}

Первый компонент каждой пары — это имя атрибута, второй компонент — имя соответствующего домена. Поэтому можно договориться, что строка заголовка действительно представляет собой заголовок. На практике заголовок отношения обычно рассматривают просто как набор имен атрибутов (то есть имена доменов часто опускаются), за исключением случаев, когда очень важна точность.

Что касается остальной части таблицы, то это, конечно, набор строк. Давайте рассмотрим одну строку, например (D2, Развития, 10M). Эта строка действительно представляет собой набор упорядоченных пар:

```
{ <DEPT#:'D2'       >,
 <DNAME:'Развития' >,
 <BUDGETS:10M      > }
```

Первый компонент каждой пары — это имя атрибута, второй компонент — соответствующее значение атрибута. Часто в неформальном описании опускают имена атрибутов, потому что известно, что каждое отдельное значение в таблице действительно является значением атрибута, имя которого находится сверху соответствующего столбца; кроме того, мы знаем, что значение принадлежит домену, который лежит в основе этого атрибута. Например, значение «D2» — это значение атрибута DEPT#, и оно взято из соответствующего домена, который лежит в его основе, а именно домена кода подразделения (который также называется DEPT#). Таким образом, можно принять, что каждая строка является кортежем в соответствии с определением.

Отношение — это достаточно абстрактный вид объекта, а таблица — это конкретное изображение (обычно, на бумаге) этого абстрактного объекта. Конечно, они очень близки и в неформальном контексте обычно их отождествляют.

Одним из неопровержимых преимуществ реляционной модели является то, что ее основной абстрактный объект — отношение, имеет простое представление на бумаге (в виде таблицы); именно такое представление делает реляционные системы простыми в использовании и простыми для понимания, а также облегчает понимание их поведения. Однако, к сожалению, табличное представление предусматривает некоторые неверные факты. Например, оно явно предусматривает, что строки таблицы (то есть кортежи отношения) расположены в определенном порядке сверху вниз, хотя это неверно.

Для упрощения записи часто применяют такую форму описания реляционной БД: Имя_таблицы (Список атрибутов).

Пример: Сотрудники (ТабНомер, Фамилия, Имя, Отчество, Должность, Оклад).

Ключевое поле обычно подчеркивают.

1.2 Общие сведения о СУБД MySQL

Разработчиком MySQL, одной из популярных систем управления базами данных (СУБД) с открытым кодом, является компания MySQL AB. В 2008 году Sun Microsystems приобрела эту компанию, а после поглощения в 2009 году Sun Microsystems компанией Oracle Corporation MySQL стала собственностью Oracle.

Открытость исходного кода означает, что любой желающий имеет возможность использовать и модифицировать это программное обеспечение по своему усмотрению. Получить и развернуть программное обеспечение MySQL можно с сайта разработчика (`www.mysql.com`), причем абсолютно бесплатно. Программное обеспечение MySQL распространяется по лицензии GPL (GNU General Public License), которая регламентирует, что разрешено, а что нет в отношении программного обеспечения.

Изначально сервер MySQL был разработан для более быстрого управления большими базами данных и в течение нескольких лет он успешно эксплуатировался в средах, к которым предъявлялись достаточно высокие требования. Невзирая на то, что MySQL находится в процессе постоянной разработки, на сегодняшний день он предоставляет богатый набор удобных в эксплуатации средств и функций. Возможности сетевого взаимодействия, производительность и безопасность, что присущие серверу MySQL, делают его удачным вариантом для работы с базами данных в Internet.

СУБД MySQL является клиент-серверной системой, которая включает многопоточный SQL-сервер, поддерживающий разные платформы, несколько клиентских программ и библиотек, инструменты администрирования и широкий диапазон программных интерфейсов и дополнений. Сервер MySQL существует также в формате встраиваемой многопоточной библиотеки, которую можно связать с разрабатываемыми приложениями, для получения компактного, быстрого и легкого в управлении продукта.

На новый уровень СУБД MySQL вышла с выпуском версии 5.0, в которой был значительно расширен набор функций, ставящей MySQL в один ряд с коммерческими СУБД. С появлением пятой версии появилась практически полная поддержка стандарта SQL. MySQL 5.0 содер-

жит такие нововведения: хранимые процедуры и функции, обработчики ошибок, курсоры, триггеры, представления, информационную схему. В последующих версиях совершенствование MySQL продолжено. На момент написания настоящего практикума последней является версия 5.7.

1.3 Установка и настройка MySQL

Рассмотрим вариант установки MySQL версии 5.5.55. Для установки MySQL запустите скачанный файл пакета установки с правами администратора.

На первом шаге мастера установки необходимо нажать кнопку «Next» для перехода к следующему шагу.

На втором шаге мастера установки необходимо ознакомится с лицензионным соглашением, установить флажок «I accept the terms in the License Agreement» (я принимаю условия лицензионного соглашения) и с помощью кнопки «Next» перейти к следующему шагу.

На третьем шаге мастера установки необходимо выбрать тип установки. Для задания полной конфигурации установки необходимо выбрать тип установки «Custom» (выборочная) с помощью кнопки «Custom».

На четвертом шаге мастера необходимо выбрать компоненты для установки, оставляем значения по умолчанию, и с помощью кнопки «Next» перейти к следующему шагу.

На шестом шаге мастера установки необходимо нажать кнопку «Install» (установить) для начала процесса установки, после чего начнется процесс установки.

По завершении процесса установки мастер установки предложит запустить мастер конфигурации экземпляра MySQL сервера, для чего отмечаем флажок «Launch the MySQL Instance Configuration Wizard» (запустить мастер конфигурации экземпляра MySQL) и нажимаем кнопку «Finish».

После выполненных действий запускается мастер конфигурации экземпляра MySQL сервера. На первом шаге необходимо нажать кнопку «Next» для перехода к следующему шагу.

На втором шаге мастера конфигурации необходимо указать тип конфигурации, устанавливаем переключатель «Detailed Configuration»

(детальная конфигурация) и с помощью кнопки «Next» перейти к следующему шагу.

На третьем шаге мастера конфигурации необходимо выбрать тип настраиваемого экземпляра MySQL сервера. Выбираем переключатель «Developer Machine» (компьютер разработчика), и с помощью кнопки «Next» перейти к следующему шагу. При таком варианте MySQL сервер будет использовать минимум оперативной памяти.

На четвертом шаге мастера конфигурации необходимо указать область применения баз данных, которыми будет управлять экземпляр MySQL сервера, выбираем переключатель «Multifunctional Database» (многофункциональная база данных) и с помощью кнопки «Next» перейти к следующему шагу.

На пятом шаге мастера конфигурации необходимо указать расположение файла InnoDB (можно оставить значение по умолчанию) и с помощью кнопки «Next» перейти к следующему шагу мастера конфигурации.

На шестом шаге мастера конфигурации необходимо определить количество параллельных подключений (например, «Decision Support» – не более 20 подключений) и с помощью кнопки «Next» перейти к следующему шагу.

На седьмом шаге мастера конфигурации необходимо задать порт, используемый для подключений к экземпляру MySQL сервера, оставьте порт по умолчанию «3306», а так же установите флажок «Add firewall exception for this port» (добавить исключения в межсетевой экран для данного порта) и с помощью кнопки «Next» перейти к следующему шагу.

На восьмом шаге мастера конфигурации следует задать кодовую страницу, используемую экземпляром MySQL–сервера по умолчанию, для чего необходимо установить переключатель «Manual Select Default Character Set/Collation» (пользовательская установка кодовой страницы) и из раскрывающегося списка «Character Set» выбрать «utf8», после чего с помощью кнопки «Next» перейти к следующему шагу.

На девятом шаге мастера конфигурации необходимо задать тип запуска MySQL сервера выбираем вариант запуска «Install As Windows Service» (установить как службу Windows) и установить флажок «Include Bin Directory in Windows PATH», что добавит путь установки

MySQL сервера в системную переменную PATH. При установленном флажке «Launch the MySQL Server automatically» MySQL сервер будет запускаться в виде службы каждый раз при загрузке Windows. После чего с помощью кнопки «Next» перейти к следующему шагу.

На десятом шаге мастера конфигурации необходимо задать пароль для пользователя root и с помощью кнопки «Next» перейти к следующему шагу.

На одиннадцатом шаге мастера конфигурации все этапы настройки завершены, необходимо нажать кнопку «Execute» (выполнить) после чего внесенные настройки применяются к экземпляру MySQL сервера.

Завершается конфигурирование нажатием кнопки «Finish» (завершить).

1.4 Оболочка dbForge Studio for MySQL

dbForge Studio for MySQL — гибкий профессиональный инструмент для разработчиков баз данных и пользователей MySQL, разработанный компанией Devart. Он дает возможность разрабатывать SQL-хранимые процедуры и функции, создавать и выполнять запросы, редактировать данные, осуществлять их экспорт и импорт, управлять пользователями, редактировать объекты баз данных, работать с проектами баз данных и т.п.

Скачать программу можно с сайта `www.devart.com`.

На рисунке 1.3 представлен внешний вид dbForge Studio for MySQL.

Строка главного меню состоит из 8 пунктов.

Пункт меню «Файл» позволяет осуществлять стандартные операции по созданию объектов (проекта, SQL запроса, диаграммы, файла), открытию существующих, сохранению, закрытию, а также выходу из программы.

Пункт меню «Правка» предоставляет возможности по редактированию текста (копирование, вставка, поиск), а также управлению автозаполнением кода.

Пункт меню «Вид» позволяет отображать и скрывать отдельные элементы окна программы (проводник баз данных, окно свойств и т.п.).

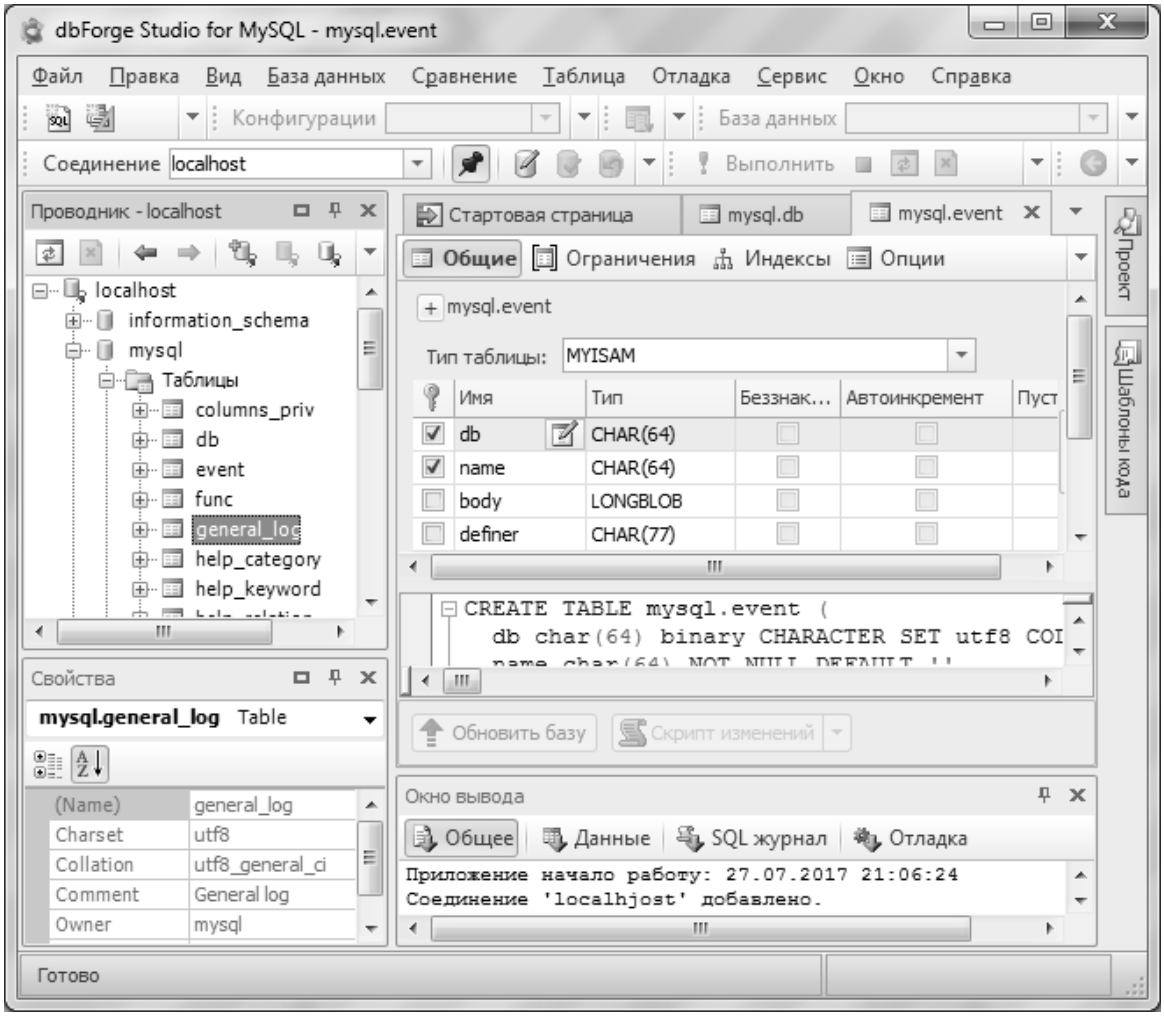

Рисунок 1.3 — Окно программы dbForge Studio for MySQL

Для управления базами данных и их элементами используется пункт меню «База данных». Назначение команд этого меню представлено в таблице 1.1.

Управление проектами баз данных осуществляется с помощью пункта главного меню «Проект». Пункт «Отладка» позволяет выполнять пошаговую отладку SQL-выражений и процедур.

Пункт меню «SQL» позволяет запускать выполнение SQL-инструкций, создавать, редактировать и удалять их параметры, а также просматривать историю выполнения SQL-инструкций.

Пункт меню «Сервис» позволяет осуществлять настройку программы и панелей инструментов.

Управление окнами осуществляется с помощью пункта меню «Окно», а просмотр справки — пункта меню «Справка».

Таблица 1.1 — Назначение команд меню «База данных»

Команда	Назначение
Создать	Создание соединения с сервером, базы данных, объектов базы данных, а также сравнений схем и данных
Начать транзакцию Фиксировать, откат	Управление ходом выполнения транзакций
Менеджер безопасности	Создание пользователей и предоставление им преимущество работы с базами данных и их объектами
Менеджер сессий	Отображает информацию о параметрах сессий пользователей
Переменные сервера	Просмотр и экспорт системных и статусных переменных сервера
Управление службами	Запуск и остановка работы служб MySQL
Найти объекты	Поиск объектов баз данных
Экспорт схемы	Экспорт схемы базы данных в файл `*.sql`
Импорт схемы	Импорт схемы базы данных из файла `*.sql`
Обслуживание таблиц	Выполнение анализа, проверки, оптимизации, восстановления таблиц баз данных
Записать кэш	Запись на диск таблиц, журналов, статусов, привилегий и других параметров
Генерация DDL	Автоматическое создание скрипта создания объектов баз данных
Выполнить скрипт	Выполнение сценария из файла `*.sql`

Панели инструментов дают доступ к большинству функциональных возможностей программы.

В «Проводнике баз данных» отражаются все соединения с сервером, перечень баз данных и их объектов. Проводник позволяет управлять объектами с помощью контекстного меню.

В окне настроек отображаются настройки выбранного соединения, базы данных или ее объекта.

Документы отображаются в виде вкладок, кроме того документы можно просматривать в различных режимах: например, таблицу — в режиме дизайна, в режиме данных или SQL.

В нижней части окна программы расположено окно вывода, в котором выводятся результаты выполнения SQL-инструкций, в том числе

данные, полученные в результате выполнения запросов, а также ошибки, возникающие в ходе выполнения.

Для того чтобы начать работу с MySQL, необходимо установить соединение с сервером. Для этого в «Проводнике баз данных» необходимо нажать кнопку «Новое соединение», после чего появится окно, изображенное на рисунке 1.4.

В данном диалоговом окне необходимо задать следующие параметры:

– хост (IP-адрес или имя сервера, к которому выполняется подключение);

– порт (сетевой порт, используемый MySQL-сервером, заданный при установке, по умолчанию 3306);

– имя (имя пользователя, зарегистрированного в СУБД);

– пароль (пароль пользователя);

– имя соединения (название вашего соединения).

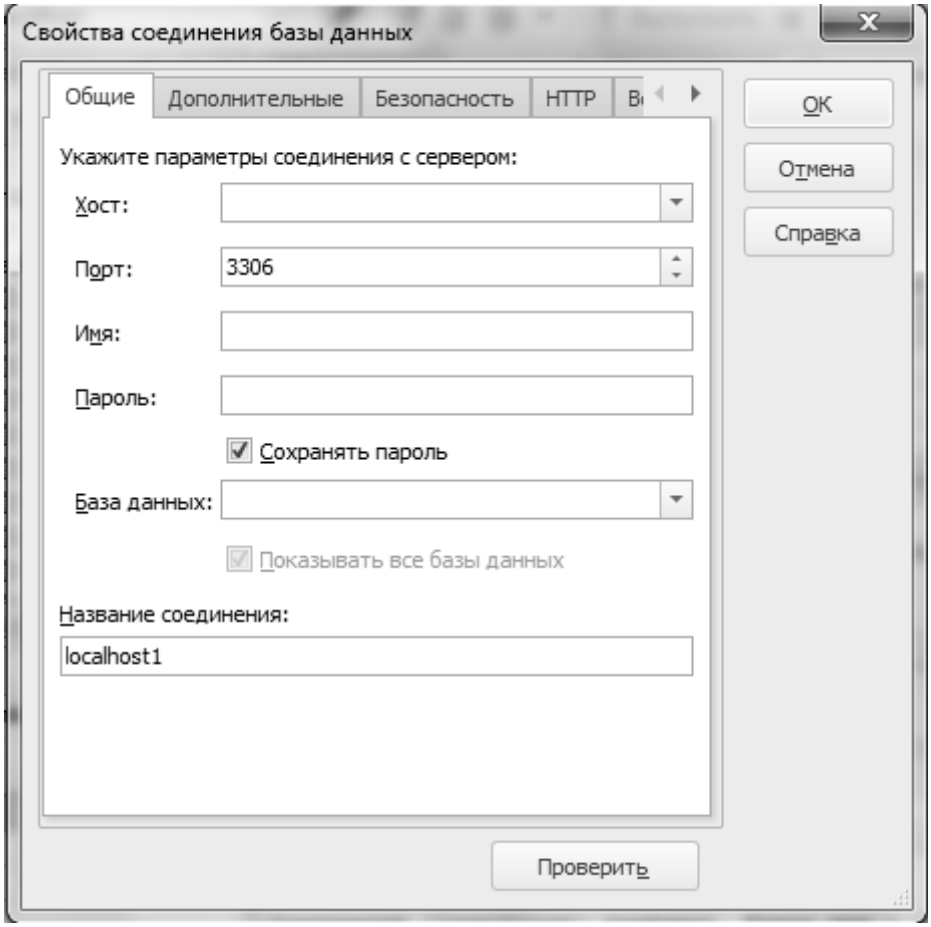

Рисунок 1.4 — Окно соединения с БД программы dbForge Studio for MySQL

Для выполнения SQL инструкций в среде dbForge Studio for MySQL необходимо создать новый SQL с помощью меню «Файл → Создать → SQL» или кнопки на панели инструментов.

1.5 Основные сведения о языке запросов SQL

SQL (Structured Query Language) — структурированный язык запросов, стандартный язык запросов для работы с реляционными базами данных. Был разработан в конце 1970-х годов в компании IBM Research и реализован в первом прототипе реляционной СУБД IBM System R. В дальнейшем этот язык применялся во многих коммерческих СУБД и постепенно де-факто стал стандартом для языков манипулирования данными в реляционных СУБД.

SQL нельзя в полной мере отнести к традиционным языкам программирования, т.к. он не содержит операторов, управляющих ходом выполнения программы, операторов описания типов и др. SQL включает только набор стандартных операторов доступа к данным, хранящимся в базе данных. Операторы SQL легко встраиваются в программы, написанные на языках программирования: C++, C#, Visual Basic, Delphi, и др. Кроме того, операторы SQL могут выполняться непосредственно в интерактивном режиме. Язык SQL содержит в себе два основных компонента:

– язык DDL (Data Definition Language) для определения структуры базы данных;

– язык DML (Data Manipulation Language) для манипулирования данными.

Первый официальный стандарт языка SQL был принят в 1986 году и уточнен в 1989 году. Дальнейшее развитие SQL поставщиками СУБД потребовало принятия в 1992 году нового расширенного стандарта SQL92 (SQL2). Следующим стандартом стал SQL:1999 (SQL3). В нем была добавлена поддержка регулярных выражений, рекурсивных запросов, триггеров, базовые процедурные расширения, не скалярные типы данных и некоторые объектно-ориентированные возможности. В настоящее время действует стандарт, принятый в 2003 году (SQL:2003), в котором введены расширения для работы с XML-данными, функции для работы с OLAP-базами данных, генераторы последовательностей и ос-

нованные на них типы данных. В 2006 и 2008 годах в стандарт SQL:2003 были внесены небольшие модификации.

1.6 Работа с базами данных

При создании базы данных в каталог данных MySQL автоматически добавляется соответствующий каталог. Все таблицы, добавляемые в базу данных, отображаются в виде файлов в этом каталоге.

Создание базы данных с помощью SQL осуществляется с помощью оператора `CREATE DATABASE:`

```
CREATE DATABASE [IF NOT EXISTS] <имя_базы_данных>
  [[DEFAULT] CHARACTER SET <имя_набора_символов>]
  [[DEFAULT] COLLATE <имя_сопоставления>]
```

Если база данных с таким именем уже существует, и конструкция `IF NOT EXISTS` не используется, сервер MySQL вернет ошибку, а если конструкция `IF NOT EXISTS` указана, то сервер вернет предупреждение. В результате новая база данных создана не будет. Конструкция `CHARACTER SET` указывает набор символов, который должен использоваться для новой базы данных по умолчанию, а конструкция `COLLATE` — тип сопоставления. Сопоставление определяет способ сравнения, сортировки и группировки значений, созданных с помощью определенного набора символов. Следующий пример создает базу данных `service` и задает набор символов — кириллицу.

```
CREATE DATABASE IF NOT EXISTS service
  CHARACTER SET utf8
  COLLATE utf8_general_ci;
```

Чтобы удалить базу данных из системы, нужно просто выполнить оператор `DROP DATABASE`:

```
DROP DATABASE [IF EXISTS] <имя_базы_данных>
```

Внимание! Когда удаляется база данных, также удаляются все таблицы и все данные, содержащиеся в этих таблицах.

В dbForge Studio for MySQL создание баз данных выполняется с помощью команды меню «База данных → Создать → Базу данных» или контекстного меню созданного соединения с сервером в «Проводнике баз данных». При создании БД необходимо ввести ее имя, набор символов и тип сопоставления.

1.7 Создание таблицы средствами SQL

Создание таблицы осуществляется с помощью оператора `CREATE TABLE:`

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] <имя_таблицы>
  (<имя_поля1> тип [NOT NULL | NULL]
  [AUTO_INCREMENT] [PRIMARY] KEY]
  [, <имя_поля2> тип [NOT NULL | NULL]
  [AUTO_INCREMENT] [PRIMARY] KEY]) , …]
)
```

Параметр `TEMPORARY` означает, что создается временная таблица (она существует только до окончания сеанса или до явного ее удаления).

Параметр `IF NOT EXISTS` имеет то же значение, что и в операторе `CREATE DATABASE`.

Поле может иметь один из типов, поддерживаемых в MySQL: числовой, строковый, двоичный или дата/время.

Числовые типы данных определяются для полей, которые не будут содержать ничего кроме цифр. Они делятся на 2 группы: целочисленные и вещественные. Синтаксис записи целочисленного типа данных:

```
TINYINT|SMALLINT|MEDIUMINT|INT|INTEGER|
BIGINT [(длина)] [UNSIGNED] [ZEROFILL]
```

Характеристики целочисленных типов, приведены в таблице 1.2.

Таблица 1.2 — Целочисленные типы данных MySQL

Тип данных	Допустимые значения	Объем памяти
TINYINT	Со знаком: от -128 до 127, без знака от 0 до 255	1 байт
SMALLINT	Со знаком: от -32768 до 32767, без знака от 0 до 65535	2 байта
MEDIUMINT	Со знаком: от -8388608 до 8388607, без знака от 0 до 16777215	3 байта
INT, INTE-GER	Со знаком: от -2147483648 до 2147483647, без знака от 0 до 4294967295	4 байта
BIGINT	Со знаком: от -9223372036854775808 до 9223372036854775807, без знака от 0 до 18446744073709551615	8 байт

Ключевое слово `UNSIGNED` означит, что в данном поле не могут храниться отрицательные значения. Когда указывается ключевое слово `ZEROFILL`, в начале вводимого в поле значения, будут добавляться нули, если количество символов в этом значении не соответствует числу, введенному в параметре <длина>. Например, если значение <длина> равно 4, то число 53 будет отображаться как 0053.

Синтаксис описания вещественных типов данных следующий:

```
FLOAT | DOUBLE | DOUBLE PRECISION | REAL
   | DECIMAL | DEC | NUMERIC | FIXED
   [(<длина>, <десятичные_разряды>)]
[UNSIGNED] [ZEROFILL]
```

Описание вещественных типов данных приведено в таблице 1.3.

Таблица 1.3 — Вещественные типы данных

Тип данных (псевдоним)	Описание
FLOAT	Со знаком: от -3,4E+38 до -1,17E-38; 0; от 1,17E-38 до 3,4E+38
DOUBLE (DOUBLE PRECI-SION, REAL)	Со знаком: от -1,8E+308 до -2,27E-308; 0; от 2,2E-308 до 1,8E+308
DECIMAL (DEC, NUMERIC, FIXED)	Точный тип данных, объем памяти для которого определяется значениями <дли-на> и <десятичные_разряды>

Строковые типы чаще всего применяют для хранения значений, которые могут включать буквы и цифры.

В MySQL существует 2 строковых типа данных: CHAR и VARCHAR. Их синтаксис приведен ниже:

```
CHAR (<длина>) [BINARY | ASCII | UNICODE]
VARCHAR (<длина>) [BINARY]
```

Ключевые слова `BINARY, ASCII, UNICODE` имеют следующее значение:

– `BINARY` — при выполнении сортировки или операций сравнения учитывался регистр символов;

– `ASCII` – присваивает полю набор символов latin1;

– `UNICODE` — присваивает полю набор символов usc2.

Тип `CHAR` — это символьный тип фиксированной длины, может хранить до `255` символов. Параметр `<длина>` определяет, какое количество символов должно сохраняться. Хотя фактическое значение может составлять и меньше, чем указано, объем памяти, который выделяется является фиксированным и равна значению параметра `<длина>`. Тип `VARCHAR` также позволяет задавать максимальную длину, но в этом случае объем памяти, который выделяется уже будет зависеть от фактического количества символов.

Кроме строковых типов данных MySQL поддерживает 4 двоичных типа данных (таблица 1.4).

Таблица 1.4 — Двоичные типы данных

Тип данных	**Максимальный размер**	
TINYBLOB, TINYTEXT	До 255 символов	355 байт
BLOB, TEXT	До 65535	64 Кбайт
MEDIUMBLOB,MEDIUMTEXT	До 16777215	16 Мбайт
LONGBLOB, LONGTEXT	До 4294967295	4 Гбайт

Типы даты и времени позволяют определять поля, которые будут содержать только данные дат и времени (таблица 1.5). Здесь Г — год, М — месяц, Д — день.

Таблица 1.5 — Типы данных даты и времени

Тип данных	Формат	Диапазон
DATE	ГГГГ-ММ-ДД	1000-01-01 до 9999-01-01
TIME	ЧЧ:ММ:СС	-838:59:59 до 838:59:59
DATETIME	ГГГГ-ММ-ДД ЧЧ:ММ:СС	1000-01-01 00:00:00 до 9999-01-01 00:00:00
YEAR	ГГГГ	От 1901 до 2155
TIMESTAMP	ГГГГ-ММ-ДД ЧЧ:ММ:СС	1970-01-01 00:00:00 частично до 2037-01-01 00:00:00

Кроме типа данных, для каждого поля в операторе `CREATE TABLE` может быть задан набор необязательных параметров:

– `NOT NULL` — недопустимость пустых значений;

– `NULL` — допустимость пустых значений;

– `AUTO_INCREMENT` — автоматическое генерирование значений поля путем увеличения на 1 максимального среди всех значений поля;

– `PRIMARY KEY` — поле выступает в качестве первичного ключа.

Поля таблицы перечисляются через запятую.

Например, создадим таблицу `workers`, содержащую информацию о работниках:

```
CREATE TABLE workers (
  IDWorker int(11) NOT NULL AUTO_INCREMENT,
  FIO varchar(80),
  birthday date,
  gender tinyint(1) UNSIGNED DEFAULT 0,
  address varchar(80) DEFAULT NULL,
  phone varchar (15) DEFAULT NULL,
  PRIMARY KEY (IDWorker)
);
```

Наша таблица содержит следующие поля:

– `IDWorker` — первичный ключ, который не может иметь пустых значений и заполняется автоматически;

– `FIO` — фамилия, имя, отчество работника;

– `birthday` — дата рождения работника;

– `gender` — пол работника (например, 0 — мужчина, 1 — женщина);

– `address` — домашний адрес работника;

– phone — телефон.

Для создания внешнего ключа используются ключевые слова CONSTRAINT, FOREIGN KEY и REFERENCES:

```
CONSTRAINT <имя_ограничения> FOREIGN KEY
  (<имя_поля>[, ...])
  REFERENCES <имя_таблицы> (<имя_поля>[, ...])
    [ON DELETE <действия>]
    [ON UPDATE <действия>]
```

Здесь <имя_таблицы>(<имя_поля>) — имя первичного ключа и таблицы, в которой он содержится.

Необязательные конструкции ON DELETE и ON UPDATE указывают, как поступать с данными в подчиненной таблице в случае удаления (обновления) строки в родительской таблице. Для каждой из конструкций доступно 5 действий, но использовать можно только одно. Значения параметров приведены в таблице 1.6.

Таблица 1.6 — Параметры конструкций ON DELETE и ON UPDATE

Опция	Описание
RESTRICT	Строки в родительской таблице не удаляются, но и не обновляются, если есть строки с соответствующими значениями ключа в дочерней таблице
CASCADE	Строки в дочерней таблице удаляются (обновляются)
SET NULL	Значение внешних ключей в дочерней таблице устанавливаются в NULL
NO ACTION	В дочерней таблице ничего не происходит
SET DEFAULT	Значение внешних ключей в дочерней таблице устанавливаются в значение по умолчанию

Создадим таблицу orders, которая содержит внешний ключ id_workers_fk:

```
CREATE TABLE orders (
  IDOrder int(11) NOT NULL AUTO_INCREMENT,
  date_begin date DEFAULT NULL,
  date_end datetime DEFAULT NULL,
  IDWorker int(11) DEFAULT NULL,
  payment tinyint(1) DEFAULT NULL,
```

```
  PRIMARY KEY (IDOrder),
  CONSTRAINT id_workers_fk FOREIGN KEY (IDWorker)
    REFERENCES workers (IDWorker) ON UPDATE CASCADE
)
```

1.8 Создание таблиц в среде dbForge Studio for MySQL

Создать таблицу можно: вызвав контекстное меню объекта «Таблицы» в «Проводнике баз данных» и выбрав команду «Новая таблица», или с помощью кнопки на панели инструментов, или в меню «База данных → Создать → Объект базы данных → Таблица». Создадим новую таблицу `orders` (рисунок 1.5).

После нажатия кнопки «Создать» открывается окно «Новый объект», в котором таблицу можно увидеть в трех режимах: дизайн, текст, данные. В режиме дизайн на вкладке «Общее» поля добавляются в таблицу нажатием клавиши `Insert` на клавиатуре или с помощью пункта «Новый столбец» контекстного меню окна таблицы. Необходимо указать имя поля, тип данных, допускаются ли пустые значения, является ли поле первичным ключом и другие настройки. Создадим первичный ключ `IDOrder` (рис. 1.6).

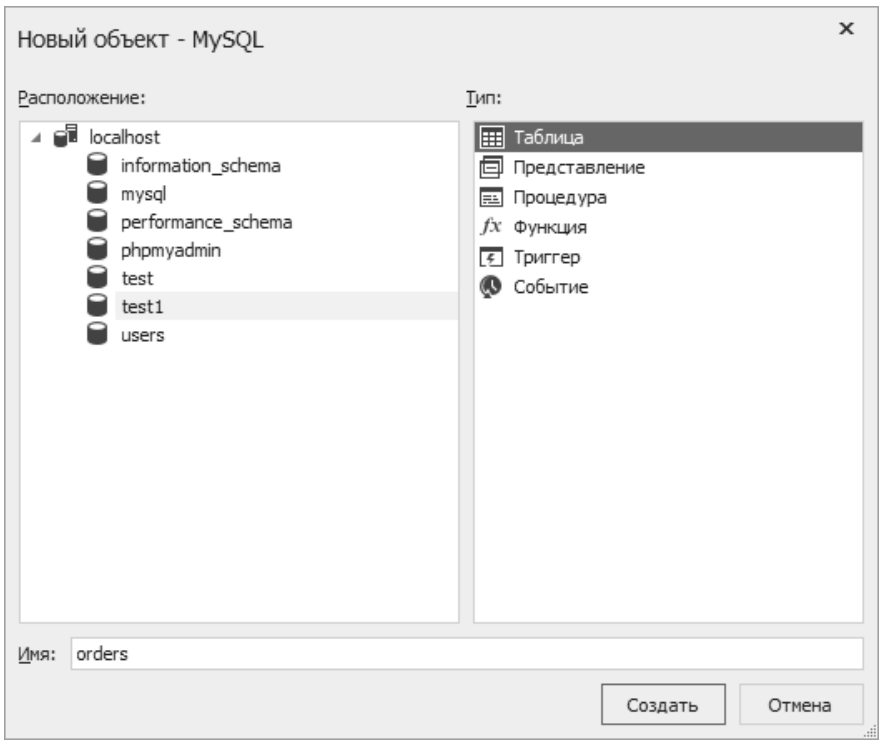

Рисунок 1.5 — Создание новой таблицы

Аналогично создадим остальные поля. Если таблица содержит внешний ключ, то его можно задать на вкладке «Ограничения» (клавиша `Insert` для добавления нового внешнего ключа), указав, какое поле таблицы является внешним ключом, и с которым полем которой таблицы оно связано. В нашем случае внешним ключом является `IDWorker`, связанный с первичным ключом `IDWorker` таблицы `workers` (рис. 1.7).

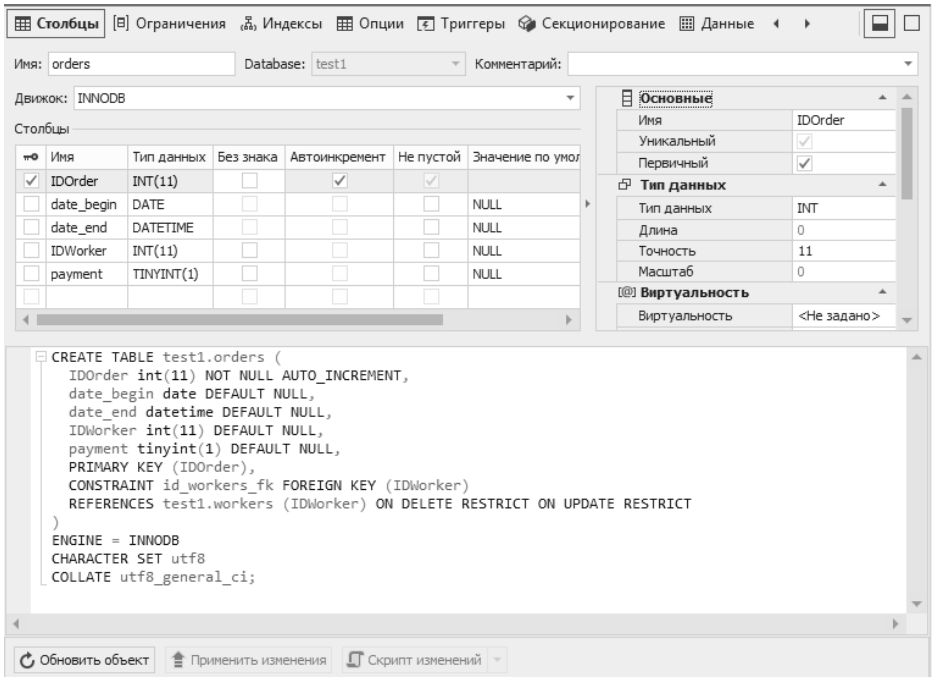

Рисунок 1.6 — Создание первичного ключа

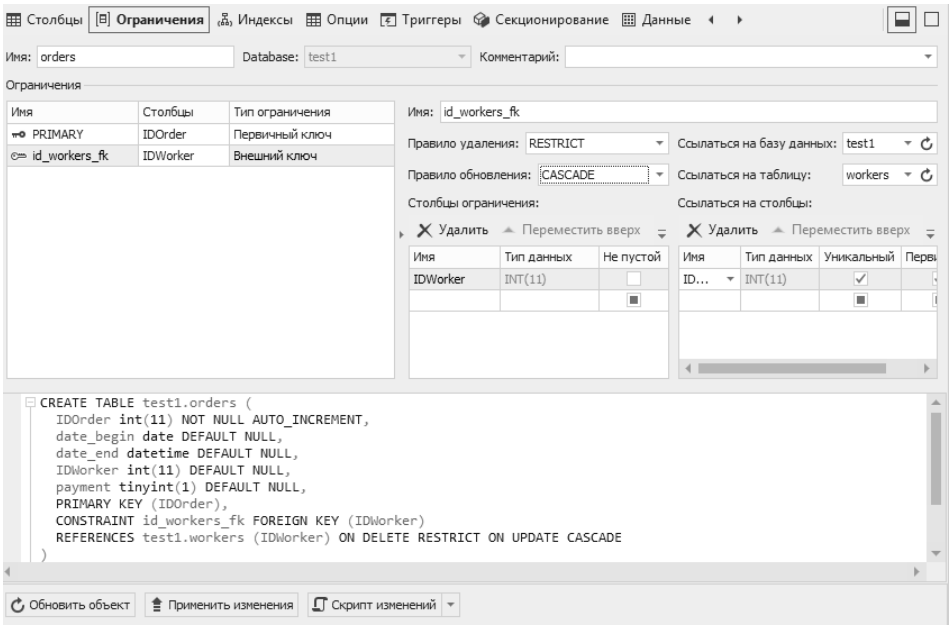

Рисунок 1.7 — Установка внешнего ключа

Также можно выбрать действия для дочерней таблицы при удалении или обновлении записей родительской (опции ON DELETE и ON UPDATE).

Обратите внимание, что созданный первичный ключ автоматически добавляется на вкладку «Ограничения».

1.9 Изменение и удаление таблиц

Часто возникает необходимость внести изменения в таблицу уже после ее создания. Для этого используется оператор ALTER TABLE:

```
ALTER TABLE <имя_таблицы>
  {ADD [COLUMN] <имя_поля> тип
    [NOT NULL | NULL] [AUTOINCREMENT]
    [PRIMARY] KEY] [FIRST | AFTER <имя_поля>]}
  | {ADD PRIMARY KEY <имя_поля>[, <имя_поля> …]}
  | {ADD FOREIGN KEY <имя_поля> [, <имя_поля> …]
     REFERENCES <имя_таблицы> [(<имя_поля>
       [{, <имя_поля>} …])]}
  | {ADD INDEX [<имя_индекса>] (<имя_поля>
      [{, <имя_поля>} …])}
| {DROP [COLUMN] <имя_поля>}
| {DROP PRIMARY KEY}
| {DROP INDEX <имя_индекса>}
```

Можно использовать несколько опций оператора ALTER TABLE сразу, например, следующий оператор удалит первичный ключ из таблицы workers и добавит новое поле remark:

```
ALTER TABLE workers
    DROP PRIMARY KEY
  ,  ADD COLUMN remark VARCHAR(255) NULL;
```

Для удаления таблицы используйте оператор:

```
DROP [TEMPORARY] TABLE [IF EXISTS]
   <имя_таблицы> [{, <имя_таблицы>} …]
```

В dbForge Studio for MySQL удаления таблицы осуществляется в Проводнике баз данных путем выбора в контекстном меню таблицы пункта «Удалить».

1.10 Работа с индексами

Индекс — это средство, используемое СУБД для ускорения операций поиска и сокращения количества времени, необходимого для выполнения сложных запросов. Индекс работает по таким же принципам, как и предметный указатель. Индекс предоставляет упорядоченный список указателей на фактические данные. Поэтому, выполняя запрос, MySQL может не сканировать каждую таблицу полностью для поиска данных, а просто проверить индекс, что сделает поиск более быстрым и эффективным.

Однако индексы имеют и свои недостатки. Во-первых, они могут влиять на скорость выполнения операций, предусматривающих внесение изменений в данные таблицы, так индекс должен обновляться каждый раз при обновлении таблицы. Кроме того, индексы занимают дополнительное дисковое пространство, что для больших таблиц может составлять значительное количество дисковой памяти.

MySQL поддерживает пять типов индексов, которые могут быть созданы в таблице (таблица 1.7).

Для создания индексов в MySQL существует несколько способов:

– при создании таблицы оператором `CREATE TABLE`;

– при изменении таблицы оператором `ALTER TABLE`;

– с помощью команды `CREATE INDEX`.

Создавая таблицу с помощью оператора `CREATE TABLE`, мы можем определить первичные и внешние ключи в описании полей. Каждый раз, когда мы создаем первичный или внешний ключ, независимо от метода, мы автоматически создаем индекс для указанных в этом ключе полей.

Чтобы создать уникальный индекс, нужно использовать ключевое слово `UNIQUE` в операторе `CREATE TABLE`:

```
[CONSTRAINT <имя_ограничения>] UNIQUE [INDEX]
[<имя_индекса>] (<имя_поля> [{,<имя_поля>}…])
```

Таблица 1.7 — Типы индексов

Тип индекса	Описание
Первичный ключ	Требует уникальности каждого значение в полях, на которых определен первичный ключ. Не допускает нулевых значений. Таблица может содержать только один первичный ключ
Внешний ключ	Устанавливает связь между полями в дочерней таблице, в которой определен внешний ключ, и соответствующими полями в родительской таблице
Обычный индекс	Базовый индекс, допускает дублированные и нулевые значения в полях, на которых он определен
Уникальный индекс	Требует, чтобы каждое значение или набор значений был уникальным в полях, на которых определен такой индекс. Допускает нулевые значения
Полнотекстовый индекс	Поддерживает полнотекстовый поиск значений в полях, на которых определен такой индекс. Допускает дублированные и нулевые значения. Полнотекстовый индекс можно создавать только в таблицах MyISAM и только на полях CHAR, VARCHAR, TEXT

Следовательно, мы могли при создании таблицы `workers` описать уникальный индекс, содержащий 2 поля: `FIO, birthday`:

```
CREATE TABLE workers (
  IDWorker int(11) NOT NULL AUTO_INCREMENT,
  FIO varchar(80),
  birthday date,
  gender tinyint(1) UNSIGNED DEFAULT 0,
  address varchar(80) DEFAULT NULL,
  phone varchar (15) DEFAULT NULL,
  PRIMARY KEY (IDWorker),
  UNIQUE index1 (FIO, birthday)
);
```

Иногда возникают ситуации, при которых необходимо проиндексировать поле, но при этом не требуется, чтобы значение в нем были уникальными. В таких ситуациях лучше использовать обычный индекс:

```
{INDEX | KEY} [<имя_индекса>]
  (<имя_поля> [{, <имя_поля>} …])
```

Существует возможность создать индекс и после создания таблицы. Для этого используется ключевое слово `ADD INDEX` в конструкции `ALTER TABLE`.

Оператор `CREATE INDEX` позволяет добавить в таблицу уникальные, обычные и полнотекстовые индексы:

```
CREATE [UNIQUE | FULLTEXT] INDEX <имя_индекса>
ON <имя_таблицы>(<имя_поля>[{,<имя_поля>}…])
```

MySQL предоставляет два метода удаления индекса из таблицы. Первый — оператором `ALTER TABLE`:

```
ALTER TABLE workers DROP INDEX index1;
```

Второй — с помощью команды `DROP INDEX`:

```
DROP INDEX index1 ON workers;
```

В среде dbForge Studio for MySQL индекс создается на вкладке «Индексы» в режиме просмотра «Дизайн» той таблицы, для которой создается индекс нажатием клавиши `Insert` или вызовом пункта «Новый индекс» контекстного меню. В открывшемся окне необходимо задать имя индекса, поля, входящие в него, и тип (рис. 1.8).

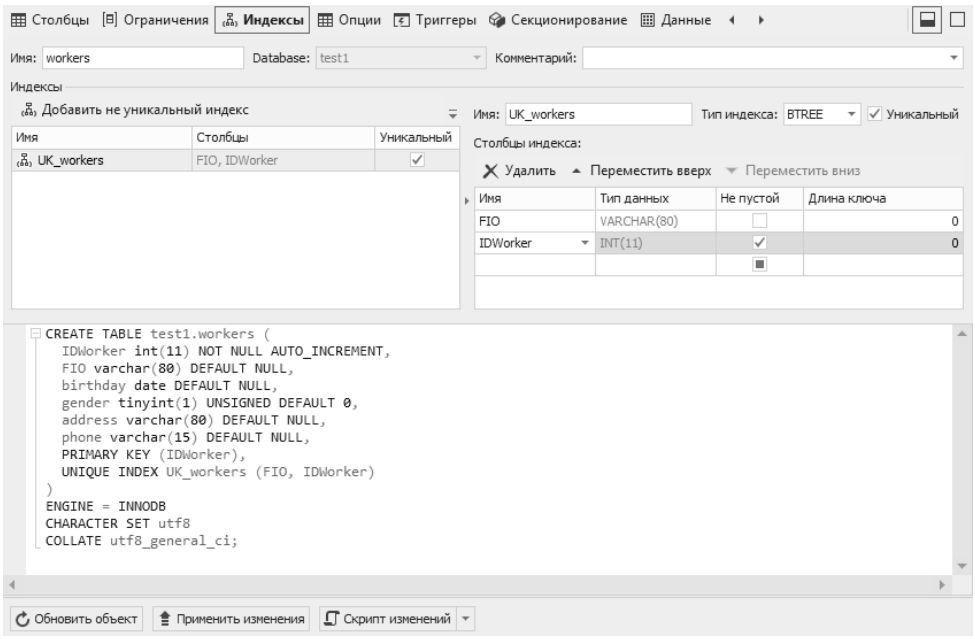

Рисунок 1.8 — Создание индекса

Удаление индекса осуществляется выбором пункта «Удалить индекс» в его контекстном меню, а изменение — пункта «Редактировать индекс» того же меню.

1.11 Основы безопасности MySQL

Одной из задач администрирования является создание и настройка учетных записей пользователей MySQL. В процессе этой настройки необходимо определить, какие пользователи будут иметь возможность подключения к серверу, откуда они смогут подключиться и что смогут делать после подключения. Для выполнения этих задач удобно использовать два оператора. Оператор `GRANT` создает пользователей MySQL и позволяет настроить их привилегии. Оператор `REVOKE` удаляет привилегии. Эти два оператора являются своего рода внешним интерфейсом для таблиц разрешений базы данных MySQL и обеспечивают альтернативу непосредственному редактированию содержимого этих таблиц. Операторы `GRANT` и `REVOKE` работают с четырьмя таблицами:

– `user` — содержит информацию о пользователях, которые подключаются к серверу и все их глобальные привилегии;

– `db` — содержит привилегии уровня базы данных;

– `tables_priv` — содержит привилегии уровня таблицы;

– `columns_priv` — содержит привилегии уровня полей.

Существует еще одна таблица разрешений (`host`), но операторы `GRANT` и `REVOKE` не в состоянии ее обрабатывать.

Если оператор `GRANT` выполняется для определенного пользователя, в таблице `user` для него создается новая запись. Если оператор определяет для пользователя любые глобальные привилегии (привилегии администратора или привилегии, применяемые сразу ко всем базам данных), они также записываются в таблицу `user`. Права обработки базы данных, таблицы или поля записываются в соответствии с таблицами `db, tables_priv и column_priv.`

Оператор `GRANT` имеет следующий синтаксис:

```
GRANT privileges (columns) ON what
  TO user IDENTIFIED BY "password" WITH GRANT OPTION
```

Здесь `privileges` — спецификатор (спецификаторы) привилегия, предоставляемых пользователю (перечислены в таблице 1.8).

Таблица 1.8 — Спецификаторы привилегий

Спецификатор	Операция, которая разрешается
USER	Пользователи, подключаемые к серверу, и все их глобальные привилегии
ALTER	Изменение таблиц и индексов
CREATE	Создание баз данных и таблиц
DELETE	Удаление существующих записей из таблиц
DROP	Удаление баз данных и таблиц
INDEX	Создание и удаление индексов
INSERT	Добавление новых записей в таблице
SELECT	Выборка существующих записей из таблицы
UPDATE	Изменение существующих записей таблиц
FILE	Чтение и запись файлов сервера
PROCESS	Просмотр информации о внутренних потоках сервера и их удаление
RELOAD	Перезагрузка таблиц разрешений или обновления журналов, кэша компьютера или кэша таблицы
SHUTDOWN	Завершение работы сервера
ALL, ALL PRIVILEGES	Все операции
USAGE	Полное отсутствие привилегий

`columns` — поля, к которым применяются определенные привилегии. Этот параметр является необязательным и используется только при задании привилегий для полей. Имена нескольких полей разделяются запятыми.

`what` — уровень применения привилегии. Привилегии могут быть глобальными (применяются ко всем базам данных и их таблицам), уровня базы данных (применяются ко всем таблицам определенной базы данных) или уровня таблицы. Используя оператор `columns`, можно определить также привилегии уровня поля.

`user` — пользователь, которому присваиваются привилегии. В некоторых версиях MySQL необходимо указывать как имя пользователя, так и компьютер, с которого он сможет подключаться. Такой способ задания легко позволяет определить двух пользователей с одинаковым именем, но которые выполняют подключения с разных компью-

теров. Возможности MySQL позволяют их различать и наделять разными правами.

password — необязательный параметр, задает пароль, присваиваемый пользователю. Если для нового пользователя опустить выражение IDENTIFIED BY пароль ему присвоен не будет. Если этот оператор задается для уже существующего пользователя, введенный пароль заменит использовавшийся до этого. Старый пароль останется неизменным, если новый не был определен. Строка пароля, задаваемая с помощью выражения IDENTIFIED BY, должна представлять собой символьную строку, оператор GRANT выполнит ее шифрование.

Оператор WITH GRANT OPTION не является обязательным. Он предоставляет пользователям право выполнять операторы GRANT, REVOKE и DROP USER (удалить пользователя). Имея привилегии GRANT, пользователь может передавать другим пользователям привилегии, которые он имеет сам.

В именах пользователей, баз данных, таблиц и паролях, записываемых в таблицу разрешений, учитывается регистр, в отличие от имен компьютеров и полей таблиц.

Например, показанный ниже оператор предоставляет возможность пользователю Reader просматривать таблицу workers базы данных.

```
GRANT SELECT ON workers
  TO Reader@localhost IDENTIFIED BY "pwd"
  WITH GRANT OPTION;
```

А следующий оператор позволяет этому же пользователю редактировать поле PhoneOwner:

```
GRANT update (FIO) ON workers
  TO Reader@localhost;
```

Пароль для пользователя задан в первом операторе GRANT, а во втором он не меняется. Кроме того, пользователь сможет предоставить другому пользователю права просматривать таблицу workers, но не

сможет предоставить привилегии для редактирования поля FIO этой же таблицы.

Для создания пользователя, который не имеет никаких прав и пароля, используется конструкция:

```
GRANT USAGE ON *.* to user_without_priv@localhost;
```

Спецификатор *.* можно условно заменить фразой «все базы данных и все таблицы». Привилегии уровня базы данных применяются ко всем таблицам определенной базы:

```
GRANT ALL ON service.*
TO Admin@MyDomain.com IDENTIFIED BY “pdw2”;
```

Этот оператор предоставляет пользователю Admin, выполняющему подключение с компьютера MyDomain.com, все права для работы со всеми таблицами базы данных service.

Для отмены привилегий пользователя применяется оператор REVOKE. Его синтаксис очень похож на синтаксис оператора GRANT с той лишь разницей, что ключевое слово TO заменено ключевым словом FROM, а ключевые слова IDENTIFIED BY и WITH GRANT OPTION вообще отсутствуют:

```
REVOKE privileges (columns) ON what FROM user;
```

Часть user должна соответствовать части user исходного оператора GRANT для пользователя, привилегии которого отменяются. Часть privileges необязательно должна соответствовать установленным ранее привилегиям. Пользуясь оператором REVOKE, можно отменить только некоторые из привилегий, предоставленных оператором GRANT.

Оператор REVOKE применяется для отмены привилегий, но не для удаления пользователей. В таблице user все равно остается запись для пользователя, даже если все привилегии для него сняты. Это означает, что пользователь все еще имеет возможность подключаться к серверу.

Для полного удаления пользователя необходимо явно удалить его запись из таблицы `user`. Для этого используется оператор `DROP USER`:

```
DROP USER user;
FLUSH PRIVILEGES;
```

Оператор `DROP USER` удаляет запись пользователя, а оператор `FLUSH` указывает серверу перезагрузить таблицы разрешений. Таблицы перезагружаются автоматически при использовании операторов `GRANT` и `REVOKE`. Однако этого не происходит при непосредственном изменении таблиц разрешений.

1.12 Управление учетными записями пользователей в среде dbForge Studio for MySQL

Оболочка dbForge Studio for MySQL предоставляет удобный интерфейс для управления пользователями и их привилегиями. Менеджер безопасности запускается из главного меню программы «База данных → Менеджер безопасности». Или командой «Редактировать привилегии» контекстного меню базы данных в Проводнике баз данных.

В левой части окна перечислены все пользователи, которые были созданы в MySQL. Для создания нового пользователя нажмите кнопку «Создать пользователя» и введите имя пользователя, пароль (рис. 1.9).

Можно указать, с какого компьютера пользователь может подключаться к серверу в поле «Хост». Введите в нем символ «`%`» или оставьте пустым, чтобы пользователь мог подключаться к серверу с любого хоста кроме `localhost`. Вы можете создать отдельную учетную запись для пользователя, если ему необходимо подключаться также с `localhost`. Хост может быть указан через его имя или IP-адрес. Здесь также можно использовать символ «%». Например, «`%.mydomain.com`» означает любой хост с домена «`mydomain.com`». Вы также можете указать максимальное число соединений, запросов и обновлений, доступное для пользователя.

После сохранения (нажмите кнопку «`Сохранить`» на панели инструментов) новый пользователь появится в списке пользователей.

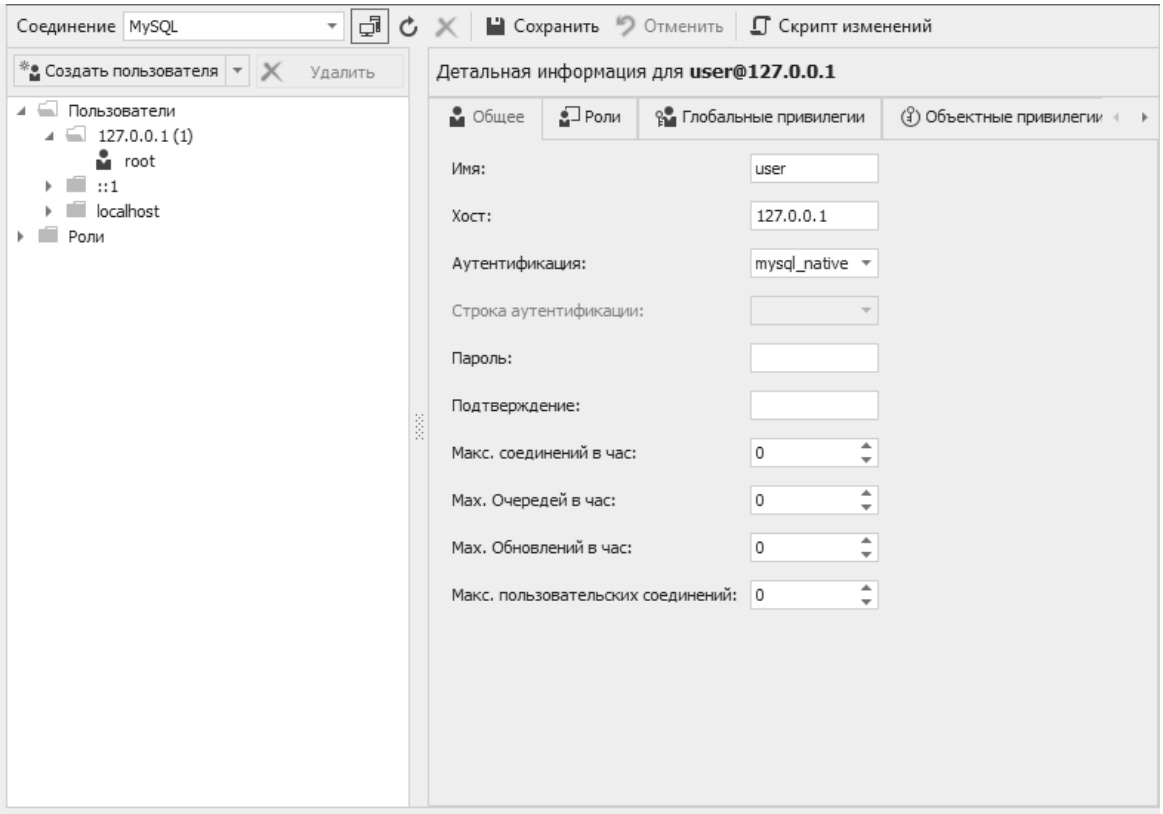

Рисунок 1.9 — Добавление нового пользователя

Для настройки прав пользователей предусмотрено 4 вкладки:

– общее (позволяет редактировать основную информацию о пользователе);

– SSL (задается метод аутентификации пользователя);

– привилегии (позволяет устанавливать глобальные привилегии);

– объекты (позволяет задавать права доступа к отдельным объектам - баз данных, таблиц, полей).

Для установления прав чтения таблицы `workers` базы данных `test1` необходимо выделить эту таблицу в списке объектов и отметить флагом привилегии `SELECT` (рис. 1.10). Автоматически в таблицу `tables_priv` будет занесено строку об установлении этой привилегии.

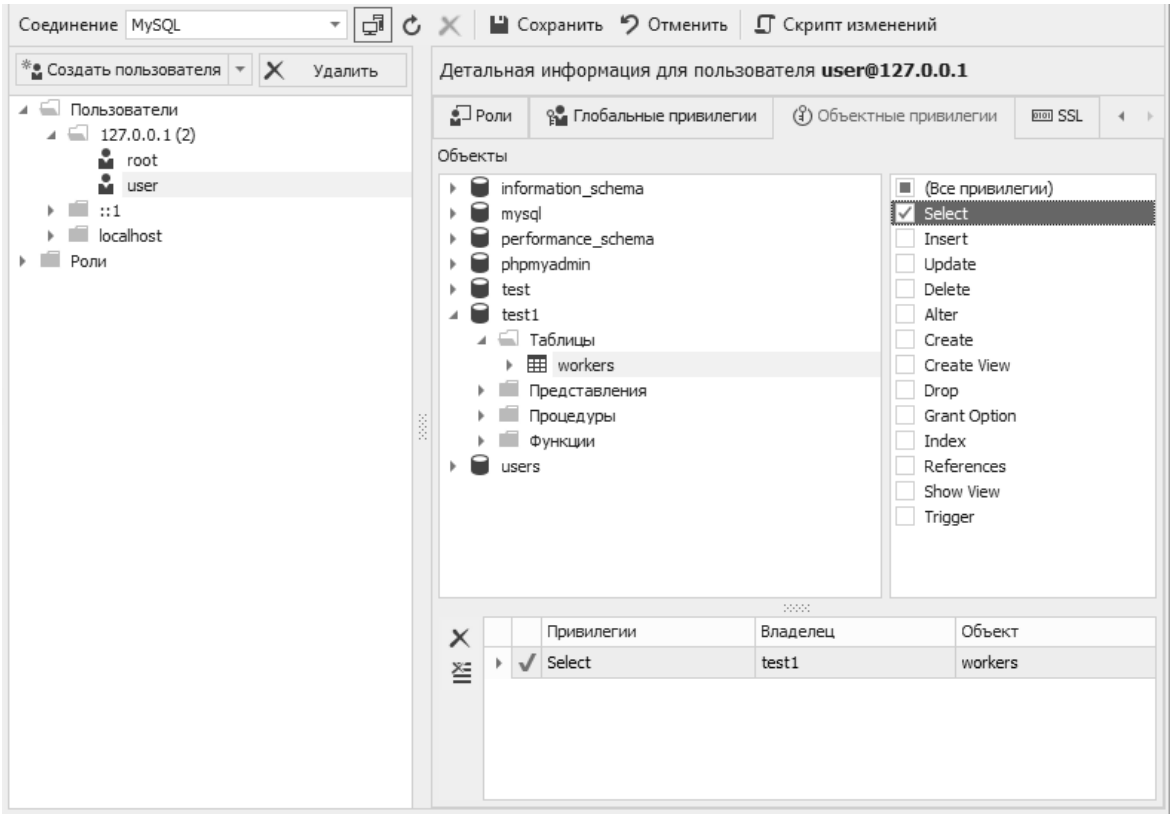

Рисунок 1.10 — Установка привилегий

1.13 Порядок выполнения работы

1. Создайте таблицы базы данных в соответствии со своим вариантом. Типы данных для полей выберите самостоятельно.

2. Создайте связи между таблицами с помощью первичных и внешних ключей.

3. Заполните таблицы данными. Количество строк в таблицах должно быть не менее количества, указанного в квадратных скобках в задании.

4. Создайте трех пользователей и предоставьте им соответствующие привилегии. Имена пользователей начинайте с названия вашей базы данных (например, `service_admin`):

– администратор базы данных — все права на управление Вашей базой данных (включая привилегии `Super`);

– аудитор (имеет возможность только просмотра всех таблиц);

– оператор справочников имеет право добавлять, удалять и редактировать строки только таблиц-справочников (содержащих только первичные ключи);

– оператор данных имеет право добавлять, удалять и редактировать строки основных таблиц (содержащих внешние ключи).

5. После выполнения всего задания создайте резервную копию базы данных.

1.14 Варианты заданий

Вариант 1. База данных банка

Сотрудники (КодСотрудника, ФИО, ДатаРождения, Пол, Адрес, Телефон, Паспорт, *Должность*) [10 записей].

Должности (КодДолжности, НазДолжности, Оклад, Обязанности) [5 записей].

ТипыВкладов (КодТипа, НазТипа, МинСрок, МинСумма, Ставка, ДопУсловия) [5 записей].

Вкладчики (КодВкладчика, ФИО, Адрес, Телефон, Паспорт) [7 записей].

Вклады (*Вкладчик*, ДатаВклада, СрокВклада, *ТипВклада*, СуммаВклада, Возвращено, *Сотрудник*) [10 записей].

Вариант 2. База данных больницы

Врачи (КодВрача, ФИО, ДатаРождения, Пол, Адрес, Телефон, Паспорт) [5 записей].

Лекарства (КодЛекарства, НазЛекарства, Показания, Противопоказания, Цена) [10 записей].

Болезни (КодБолезни, НазБолезни, Симптомы, Осложнения) [10 записей].

Рецепты (*Болезнь, Лекарства*) [15 записей].

Пациенты (КодПациента, ДатаРрождения, Пол, Адрес, Телефон) [10 записей].

Лечение (*Пациент*, ДатаОбращения, *Болезнь, Врач*, ДатаВыздоровления) [30 записей].

Вариант 3. База данных отеля

Номера (КодНомера, Номер, Вместимость, Описание, Стоимость) [10 записей].

ВидыУслуг (КодУслуги, НазУслуги, Описание, Стоимость) [5 записей].

Клиенты (КодКлиента, ФИО, ДатаРождения, Паспорт) [5 записей].

Поселение (ДатаПоселения, *Клиент, Номер*, ДатаВыезда) [30 записей].

Обслуживание (Дата, *Клиент, Услуга*) [15 записей].

Вариант 4. База данных аэропорта

Города (КодГорода, НазГорода) [5 записей].

Самолеты (КодСамолета, Марка, Вместимость, ДатаВыпуска) [10 записей].

Рейсы (КодРейса, Дата, *ГородОтправки*, *ГородНазначения, Самолет,* Продолжительность) [20 записей].

Пассажиры (КодПассажира, Фамилия, Имя, Отчество, Паспорт, Телефон) [15 записей].

ПроданныеБилеты (*Рейс, Пассажир*, Место, Цена).

Вариант 5. База данных проката дисков

Сотрудники (КодСотрудника, ФИО, ДатаРождения, Адрес, Паспорт) [5 записей].

Жанры (КодЖанра, НазЖанра, Описание) [5 записей].

Диски (КодДиска, НазваниеФильма, ГодВыпуска, *Жанр*, ЦенаЗаСутки) [15 записей].

Клиенты (КодКлиента, ФИО, Пол, Адрес, Телефон, Паспорт) [10 записей].

Прокат (*Клиент, Диск, Сотрудник*, ДатаВзятия, ДатаВозвращения, ОтметкаОбОплате) [30 записей].

Вариант 6. База данных библиотеки

Жанры (КодЖанра, НазЖанра, Описание) [5 записей].

Издательства (КодИздательства, НазИздательства, Адрес) [5 записей].

Книги (КодКниги, НазКниги, ПервыйАвтор, *Издательство*, ГодИздания, *Жанр*) [15 записей].

Читатели (КодЧитателя, ФИО, ДатаРождения, Адрес, Телефон) [10 записей].

Выдача (*Книга, Читатель*, ДатаВыдачи, ДатаВозвращения) [30 записей].

Вариант 7. База данных радиостанции

Сотрудники (КодСотрудника, ФИО, ДатаРождения, Пол, Адрес, Телефон, Паспорт) [5 записей].

Исполнители (КодИсполнителя, Название (ФИО), Описание) [10 записей].

Жанры (КодЖанра, НазЖанра, Описание) [5 записей].

Записи (КодЗаписи, НазЗаписи, *Исполнитель*, Год, *Жанр*, ДатаЗаписи, Время) [15 записей].

Вещание (Дата, Время, *Сотрудник, Запись*) [30 записей].

Вариант 8. База данных таксопарка

Водители (КодВодителя, ФИО, ДатаРождения, Пол, Адрес, Телефон, Паспорт) [10 записей].

Автомобили (КодАвто *Марка*, РегНомер, ГодВыпуска, *Водители*) [10 записей].

Марки (КодМарки, НазМарки, ТехнХаракт, Стоимость) [5 записей].

Тарифы (КодТарифа, НазТарифа, Размер) [4 записи].

Вызовы (Дата, Время, Телефон, Расстояние, *Тариф, Авто*) [30 записей].

Вариант 9. База данных туристического агентства

Отели (КодОтеля, НазваниеОтеля, Город, Адрес, Количество звезд) [5 записей].

Клиенты (КодКлиента, ФИО, ДатаРождения, Пол, Адрес, Телефон, Паспорт) [10 записей].

ДопУслуги (КодУслуги, НазУслуги, Описание, Цена) [5 записей].

Путевки (КодПутевки, ДатаПоселения, ДатаВыезда, *Гостиница*, Стоимость, *Клиент*) [30 записей]

Предоставляемые услуги (*Путевка, Услуга*, Количество) [15 записей].

Вариант 10. База данных страховой компании

Группы клиентов (КодГруппы, НазГруппы, Описание) [5 записей].

Клиенты (КодКлиента, ФИО, ДатаРождения, Пол, Адрес, Паспорт, *Группа*) [15 записей].

Филиалы (КодФилиала, НазФилиала, Адрес, Телефон) [5 записей].

ВидыПолисов (КодВида, НазВида, РазмерВзноса, СуммаВыплаты) [5 записей].

Полисы (КодПолиса, ДатаОформления, *ВидПолиса*, СрокДействия, *Клиент, Филиал*) [30 записей].

Вариант 11. База данных сервисного центра

Запчасти (КодЗапчасти, НазЗапчасти, Функции, Цена) [10 записей].

ВидыНеисправностей (КодВида, НазВида, Описание) [5 записей].

Ремонт (*Неисправность, Запчасть*, Количество) [10 записей].

Магазины (КодМагазина, НазМагазина, Адрес, Телефон) [10 записей].

Заказы (ДатаЗаказа, ДатаВозвращения, *Магазин*, СерийныйНомер, *Неисправность*, Гарантия) [30 записей].

Вариант 12. База данных транспортной компании

Водители (КодВодителя, ФИОВодителя, ДатаРождения, Адрес, Телефон) [5 записей].

МаркиАвто (КодМарки, НазМарки, ТехнХарактеристики) [5 записей].

Автомобили (КодАвто, *Марка*, РегНомер, ГодВыпуска, *Водитель*) [5 записей].

Грузы (КодГруза, НазГруза, Тариф) [15 записей].

Рейсы (Дата, Заказчик, ПунктОтправления, ПунктНазначения, *Груз*, Количество, *Авто*, Доставлено) [30 записей].

Вариант 13. База данных деканата

Студенты (НомерСтудБилета, Фамилия, Имя, Отчество, ДатаРождения) [20 записей].

Кафедры (КодКафедры, НазКафедры Факультет) [5 записей].

Преподаватели (ТабНомер, Фамилия, Имя, Отчество, *Кафедра*) [10 записей].

Предметы (КодПредмета, Название, *Преподаватель*, КоличествоЧасов) [10 записей].

Ведомость (*Студент, Предмет*, ДатаСдачи, Оценка) [35 записей].

Вариант 14. База данных строительной компании

Материалы (КодМатериала, НазваниеМатериала, Описание, Цена) [10 записей].

ВидыРабот (КодВида, НазВида, Описание, Стоимость) [10 записей].

Смета (*ВидРаботы, Материал*, Количество) [25 записей].

Заказчики (КодЗаказчика, ФИО, Адрес, Телефон, Паспорт) [10 записей].

Заказы (*Заказчик, ВидРаботы*, ДатаНачала, ДатаОкончания, ФактОплаты) [35 записей].

Вариант 15. База данных риэлтерской фирмы

Сотрудники (КодСотрудника, ФИО, ДатаРождения, Пол, Адрес, Паспорт) [5 записей].

ВидыУслуг (КодУслуги, НазУслуги, Описание, Цена) [5 записей].

ВидыКвартир (КодВида, НазВида, Описание) [5 записей].

Квартиры (КодКвартиры, *Вид*, Адрес, КоличествоКомнат, Площадь, Цена, ФИОВладельца) [15 записей].

Договоры (Дата, *Квартира, Услуга, Сотрудник*) [30 записей].

Вариант 16. База данных рекламного агентства

Сотрудники (КодСотрудника, ФИО, ДатаРождения, Пол, Адрес, Телефон) [5 записей].

Дополнительные услуги (КодУслуги, НазваниеУслуги, Описание, Стоимость) [5 записей].

Заказчики (КодЗаказчика, НазваниеЗаказчика, Адрес, Паспорт) [15 записей].

Заказы (КодЗаказа, ДатЗаказа, ДатаВыполнения, Заказчик, ФактУплаты) [20 записей].

Смета *(Заказ, Услуга, Сотрудник*) [35 записей].

Вариант 17. База данных компьютерной фирмы

ВидыКомплектующих (КодВида, НазВида, Описание) [5 записей].

Комплектующие (КодКомпл, НазваниеКомпл, *Вид*, ДатаВыпуска, Характеристики, СрокГарантии) [15 записей].

Заказчики (КодЗаказчика, Фамилия, Адрес, Телефон, Паспорт) [10 записей].

Заказы (КодЗаказа, ДатаЗаказа, ДатаВыполнения, Заказчик, ФактУплаты) [20 записей].

Спецификации (*Заказ, Комплектующие*, Количество, Цена) [35 записей].

Вариант 18. База данных ГАИ

Звание (КодЗвания, НазЗвания, Надбавка) [5 записей].

Сотрудники (КодСотрудника, ФИО, ДатаРождения, Адрес, *Звание*) [5 записей].

МаркиАвто (КодМарки, НазваниеМарки, Производитель) [5 записей].

Водители (КодВодителя, ФИО, ДатаРождения, Адрес, Паспорт, НомерВодительПрав, ДатаВыдачи, СрокДействия) [20 записей].

Автомобили (КодАвто, *Марка*, Цвет, ДатаВыпуска, *Водитель*, РегНомер, *НомерВыдал*) [30 записей].

Вариант 19. База данных кинотеатра

Страны (КодСтраны, НазваниеСтраны) [10 записей].

Жанры (КодЖанра, НазваниеЖанра, Описание) [5 записей].

Фильмы (КодФильма, Название, Режиссер, *Жанр, Страна*, Время, Ограничение) [10 записей].

Сеансы (КодСеанса, Дата, ВремяНачала, *Фильм*) [20 записей].

ПроданныеБилеты (*Сеанс*, Место, Цена) [35 записей].

Вариант 20. База данных автосалона

ТипыКузова (КодТипа, НазваниеТипа, Описание) [5 записей].

Автомобили (КодАвто, Марка, *ТипКузова*, Цвет, Год Выпуска, Цена) [30 записей].

Сотрудники (КодСотрудника, ФИО, ДатаРождения, Адрес) [5 записей].

Клиенты (КодКлиента, Фамилия, Адрес, Телефон, Паспорт) [20 записей].

Продажи (ДатаЗаказа, ДатаПродажи, *Авто, Клиент, Сотрудник*) [15 записей].

1.15 Требования к отчету

Отчет оформляется на стандартных листах белой бумаги формата А4 (210х297 мм). Текст пишется с одной стороны листа чернилами синего или черного цвета или печатается на принтере. Бланк отчета готовится во время домашней подготовки к работе и должен содержать:

– данные о студенте: фамилия и инициалы, шифр группы;

– тему и цель работы;

– дату выполнения работы;

– набор данных, необходимых для выполнения работы: задание к работе, структуру таблиц базы данных, схему базы данных, типы полей таблиц, список ограничений, данные, которые будут заноситься в таблицу, список привилегий для пользователей.

После выполнения работы к отчету подшиваются: распечатка таблиц базы данных (в виде SQL-запроса), распечатка схемы базы данных. Отчет должен быть выполнен аккуратно. Неаккуратно выполненные отчеты, а также ксерокопии чужих отчетов или их фрагментов к защите не допускаются.

1.16 Контрольные вопросы

1. Приведите определения понятий: банк данных, база данных, СУБД, реляционная база данных.

3. Приведите основные характеристики отношения.

4. Что представляет собой домен и каково его основное назначение?

5. Перечислите виды ключей в реляционной базе данных и их назначение.

6. Каково назначение языка SQL? Какой стандарт SQL дейтсвет на настоящий момент?

7. Приведите синтаксис оператора создания базы данных. Что представляют собой опции CHARACTER SET и COLLATE?

8. Что представляет собой временная таблица и для чего может понадобиться ее использование?

9. Приведите синтаксис оператора CREATE TABLE.

10. Приведите синтаксис оператора ALTER TABLE.

11. Приведите основные типы данных MySQL и их характеристики.

12. Как создать первичный ключ в таблице с помощью SQL-команд?

13. Как создать внешний ключ в таблице с помощью SQL-команд?

14. Приведите синтаксис команды для удаления объектов базы данных?

15. Дайте определение понятию индекс. Для чего он используется?

16. Какие типы индексов существуют в СУБД MySQL?

17. Способы создания индексов средствами SQL-команд.

18. В каких таблицах MySQL содержится информация о привилегиях пользователей.

19. Синтаксис оператора GRANT. Можно ли предоставить привилегии пользователю на уровне поля?

20. Синтаксис оператора REVOKE. Что делает этот оператор?

2 ЛАБОРАТОРНАЯ РАБОТА № 2

Тема: SQL-запросы, триггеры, хранимые процедуры.

Цель работы: закрепить теоретический материал по работе с SQL-запросами, триггерами, хранимыми процедурами.

2.1 Выборка данных

2.1.1 Оператор SELECT

Одной из главных функций любой реляционной системы управления базами данных является возможность получения доступа к данным, которые хранятся в БД.

Оператор SELECT — это основной оператор, используемый в MySQL для выборки данных. Синтаксис оператора SELECT состоит из ряда конструкций и других элементов, многие из которых являются необязательными, позволяющих уточнять запрос таким образом, чтобы он возвращал только необходимую информацию.

Синтаксис оператора SELECT:

```
SELECT [ALL | DISTINCT | DISTINCTROW]
  {<Список полей [AS новое имя]> | *}
FROM
  <имя таблицы [alias], [...]>
[WHERE <Фильтр для строк согласно условиям>]
[GROUP BY <Список полей результата>]
[HAVING <Фильтр для групп строк>]
[ORDER BY <Список полей сортировки результата>];
```

Ключевые слова, используемые в операторе `SELECT`, имеют следующее значение:

– `ALL` — в результат включаются все строки, удовлетворяющие условиям запроса;

– `DISTINCT` — в результат включаются все строки, удовлетворяющие условиям, кроме их дубликатов;

– `*` — в результирующий запрос включаются все столбцы из исходных таблиц;

– `alias` — сокращенное имя (псевдоним) таблицы БД;

– `FROM` — имя таблицы, из которой будет извлечены данные;

– WHERE — условие фильтрации для результирующего набора;

– GROUP BY — группировка данных при выводе;

– HAVING — дополнительное условие выборки;

– ORDER BY — сортировка результатов запроса по указанному полю (или полям).

Рассмотрим примеры запросов к базе данных. Нижеприведённый оператор SELECT выведет информацию о всех заказчиках (таблица 2.1):

```
SELECT * from customers;
```

Таблица 2.1 — Результат выполнения запроса

id_customer	name_customer	city	address
1	Сантех-люкс	Алчевск	ул. Советская 55
2	Вальмира	Донецк	ул. Советская 30
3	Акрополь	Луганск	ул. Советская 90
4	Грин Хаус	Брянка	ул. Абаканская 45
5	Планета	Алчевск	ул. Абаканская 90
6	Южные ворота	Донецк	ул. Абаканская 15
7	МираМикс	Луганск	ул. Вокзальная 20
8	Тетрис	Брянка	ул. Набережная 10
9	Инхаус	Алчевск	ул. Парковая 11
10	Меридиан	Донецк	ул. Сува 59
11	Ареал	Луганск	ул. Сува 13
12	Континент	Брянка	ул. Сува 95
13	Квартал	Брянка	ул. Ленина 66
14	Гильдия	Брянка	ул. Ленина 15
15	Белый Квадрат	Брянка	ул. Абаканская 40

Для вывода перечня городов, в которых расположены заказчики, с которыми сотрудничает фирма можно воспользоваться следующим запросом:

```
SELECT DISTINCT city AS Город FROM customers;
```

Ключевое слово DISTINCT обеспечивает отсутствие в запросе повторения городов, а конструкция AS Город устанавливает новый заголовок для вывода поля. В результате получим результат, приведенный в таблице 2.2

Таблица 2.2 — Результат выполнения запроса

Город
Алчевск
Донецк
Луганск
Брянка

Данные можно выбирать и из нескольких таблиц. Для этого предназначена операция INNER JOIN (внутреннее объединение). Она объединяет строки из двух таблиц, если связанные поля этих таблиц содержат одинаковые значения. Синтаксис:

```
FROM таблица_1
  INNER JOIN таблица_2
  ON таблица_1.поле_1 оператор таблица_2.поле_2
```

В таблице 2.3 перечислены аргументы операции INNER JOIN.

Таблица 2.3 — Аргументы операции INNER JOIN

Элемент	Описание
таблица_1, таблица_2	Имена таблиц, содержащих объединяемые строки.
поле_1, поле_2	Имена объединяемых полей. Если эти поля не являются числовыми, то должны иметь одинаковый тип данных и содержать данные одного типа, но могут иметь разные имена.
оператор	Любой оператор сравнения: «=» , «<», «>», «<=», «>=», «<>»

Операцию INNER JOIN можно использовать в любом выражении FROM. Это обычный тип связывания, он объединяет строки двух таблиц, если поля двух таблиц, имеют подходящий тип данных и содержат одинаковые данные. Попытка объединить поля двоичных типов данных приведет к возникновению ошибки.

Например: следующий запрос выведет наименования заказчиков, заключавших договора с фирмой и сумму по всем оплаченным договорам.

```
SELECT
    c.name_customer, c.city, c.address
  , SUM(o.payment) as payment
```

```
FROM
  customers c
  INNER JOIN orders o
    ON c.id_customer = o.id_customer
GROUP BY
    c.name_customer
  , c.address;
```

Результат выполнения запроса приведен в таблице 2.4

Таблица 2.4 — Результат выполнения запроса

customer	**city**	**address**	**payment**
Белый Квадрат	Брянка	ул. Абаканская 40	511,00
Вальмира	Донецк	ул. Советская 30	3200,50
Гильдия	Брянка	ул. Ленина 15	7024,00
Акрополь	Луганск	ул. Советская 90	3500,00
Ареал	Луганск	ул. Сува 13	9564,00
Грин Хаус	Брянка	ул. Абаканская 45	5750,25
Квартал	Брянка	ул. Ленина 66	6630,00
Континент	Брянка	ул. Сува 95	6019,00
Инхаус	Алчевск	ул. Парковая 11	356,00
Меридиан	Донецк	ул. Сува 59	387,00
Планета	Алчевск	ул. Абаканская 90	3650,80
МираМикс	Луганск	ул. Вокзальная 20	100,00
Сантех-люкс	Алчевск	ул. Советская 55	350,00
Тетрис	Брянка	ул. Набережная 10	6855,00
Южные ворота	Донецк	ул. Абаканская 15	3600,00

Внутреннее объединение возвращает записи из двух таблиц, если значение первичного ключа первой таблицы соответствует значению внешнего ключа второй таблицы. Однако часто возникают ситуации, когда нужно получить все строки из одной таблицы, участвующей в объединении, независимо от того, существуют ли связанные строки в другой таблице. Для этого следует использовать внешнее объединение (`OUTER JOIN`).

Запрос из предыдущего примера, но с использованием операции `LEFT JOIN`, выведет всех заказчиков независимо от того, заключали они договора с фирмой или нет:

```
SELECT
    c.customer
  , c.city
  , c.address
  , SUM(o.payment) AS payment
FROM
  customers c
  LEFT JOIN orders o
    ON c.id_customers = o.id_customers
GROUP BY
    c.name_customer
  , c.address;
```

Результат выполнения запроса приведен в таблице 2.5

Таблица 2.5 — Результат выполнения запроса

customer	**city**	**address**	**payment**
Белый Квадрат	Брянка	ул. Абаканская 40	511,00
Вальмира	Донецк	ул. Советская 30	3200,50
Гильдия	Брянка	ул. Ленина 15	7024,00
Акрополь	Луганск	ул. Советская 90	3500,00
Ареал	Луганск	ул. Сува 13	9564,00
Грин Хаус	Брянка	ул. Абаканская 45	5750,25
Квартал	Брянка	ул. Ленина 66	6630,00
Инвест Плюс	Алчевск	ул. Ленина 45	(null)
Континент	Брянка	ул. Сува 95	6019,00
Инхаус	Алчевск	ул. Парковая 11	356,00
Меридиан	Донецк	ул. Сува 59	387,00
Планета	Алчевск	ул. Абаканская 90	3650,80
МираМикс	Луганск	ул. Вокзальная 20	100,00
Проком	Брянка	ул. Победы 8	(null)
Сантех-люкс	Алчевск	ул. Советская 55	350,00
Тетрис	Брянка	ул. Набережная 10	6855,00
Южные ворота	Донецк	ул. Абаканская 15	3600,00

2.1.2 Операции в операторе SELECT

Для того чтобы компоненты выражений в операторе SELECT могли взаимодействовать друг с другом, используются операции, которые определяют тип взаимодействия или условия, ограничивающие диапазон допустимых в результирующем наборе значений. Операция — это знак или

ключевое слово, определяющее определенное действие или условие между другими элементами выражения или между выражениями.

В MySQL поддерживается ряд различных операций, которые можно разделить на пять категорий:

– арифметические операции — выполняют вычисления со значениями, указанными в выражении аргументов;

– операции сравнения — сравнивают значения аргументов в выражении для того, чтобы проверить, является ли условие истинным, ложным или NULL;

– логические операции — проверяют правильность одного или более выражений с целью узнать, что возвращает указанное в них условие: истину, ложь или NULL;

– битовые операции — обрабатывают битовые значения, которые ассоциируются с числовыми значениями;

– операции сортировки — задают сопоставление и чувствительность к регистру операций поиска и сортировки.

Арифметические операции служат для вычисления значения, указанных в выражениях аргументов. Они похожи на знаки, которые встречаются в арифметических выражениях, используемые для: сложения, вычитания, умножения или деления. MySQL поддерживает следующие арифметические операции: + (сложение), - (вычитание), - (унарный, меняет знак аргумента), / (деление), % (по модулю). Эти операции можно использовать как отдельно, так совместно. Следующий синтаксис выведет результат вычисления выражения:

```
SELECT 15 + (25 * 17 - 215);
```

В результате выполнения запроса будет вычислено число 225.

MySQL поддерживает набор операций сравнения, позволяющих определять в SQL-операторах условия различного вида. Все эти операции кратко описаны в таблице 2.6.

Логические операции позволяют проверять правильность одного или более выражений. Для того, чтобы условия считались выполненными, они должны возвращать истину. Логические операции, доступ-

ные в MySQL: AND (И), OR (ИЛИ), XOR (исключающее ИЛИ), NOT (НЕ).

Таблица 2.6 — Операции сравнения в MySQL

Операция	Описание
=	Истина, когда оба аргумента равны, только если оба аргумента не равняются NULL
<=>	Истина, когда оба аргумента равны, даже когда оба аргумента равняются NULL
<>, !=	Истина, если два аргументы не равны друг другу
<, >	Истина, если значение первого аргумента меньше (больше) значение второго аргумента
<=, >=	Истина, если значение первого аргумента меньше или равно (больше или равно) значению второго аргумента
IS NULL (IS NOT NULL)	Истина, если значение аргумента равно (не равно) NULL
BETWEEN (NOT ETWEEN)	Истина, если значение аргумента не выходит (выходит) за рамки диапазона, указанного в конструкции BETWEEN
IN (NOT IN)	Истина, если значение аргумента присутствует (не присутствует) в конструкции IN
LIKE (NOT LIKE)	Истина, если значение аргумента перечисленное (не перечисленное) в конструкции LIKE

Операции в основном применяются в предикатах WHERE и HAVING оператора SELECT, примеры их использования приведены в соответствующих разделах.

2.1.3 Предикаты WHERE и ORDER BY

Оператор SELECT выводит все строки избранных полей таблицы базы данных. На практике чаще всего встречаются задачи вывода определенных строк, удовлетворяющих некоторому условию (условиям). Для этого в операторе SELECT используется предикат WHERE, синтаксис которого выглядит так:

```
WHERE <выражение> [{<операция> <выражение>} … ]
```

Предикат WHERE должен содержать, по крайней мере, одно выражение, определяющее, какие строки возвращает оператор SELECT.

Следующий оператор выводит только заказчиков и заказы, оплата которых больше 3000:

```
SELECT
    c.customer, c.city, c.address
  , o.date_begin, o.payment
FROM
  customers c
  INNER JOIN orders o
    ON c.id_customers = o.id_customers
WHERE
  o.payment > 4000;
```

Результат выполнения запроса приведен в таблице 2.7.

Таблица 2.7 — Результат выполнения запроса

customer	**city**	**address**	**date_begin**	**payment**
Грин Хаус	Брянка	ул. Абаканская 45	07.12.2017	4800,25
Ареал	Луганск	ул. Сува 13	01.02.2018	9564,00
Квартал	Брянка	ул. Ленина 66	13.02.2018	6565,00
Гильдия	Брянка	ул. Ленина 15	05.02.2018	6688,00

Следующий запрос выводит выполняемые, в настоящее время договора (то есть поле конечной даты договора пустое — NULL):

```
SELECT
    c.customer, c.city
  , c.address , o.date_begin
  , o.date_end
FROM
  customers c
  INNER JOIN orders o
    ON c.id_customers = o.id_customers
WHERE
  o.date_end IS NULL
```

Результат выполнения запроса приведен в таблице 2.8.

Таблица 2.8 — Результат выполнения запроса

customer	city	address	date_begin	date_end
Тетрис	Брянка	ул. Набережная 10	28.11.2017	(null)
Континент	Брянка	ул. Сува 95	15.02.2018	(null)
Квартал	Брянка	ул. Ленина 66	13.02.2018	(null)
Квартал	Брянка	ул. Ленина 66	08.02.2018	(null)
Гильдия	Брянка	ул. Ленина 15	02.02.2018	(null)
Гильдия	Брянка	ул. Ленина 15	05.02.2018	(null)
Белый Квадрат	Брянка	ул. Абаканская 40	12.02.2018	(null)

Поиск значений по шаблону позволяет осуществлять операции LIKE и NOT LIKE. Выбор всех заказчиков, в названии которых встречается последовательность символов «ква»:

```
SELECT
    c. customer, c.city
FROM
  customers c
WHERE
  c. customer LIKE '%ква%'
```

Знак «%» (процент) действует как символ-заменитель. Он заменяет собой любую последовательность символов. Таким образом «%ква%» означает все названия, в которых встречается последовательность символов «ква». Аналогично «%ква» выбирает строки, в которых наименование заканчивается на «ква», а «ква%» — строки, в которых наименование начинается на «ква». Для замещения одного символа используется символ «_» (нижнее подчеркивание).

Когда в конструкции WHERE присутствует несколько условий, они объединяются с помощью операций AND или OR. Например:

```
SELECT
    c.name_customer, c.city
  , c.address,       o.date_begin
  , o.date_end,      o.payment
FROM
  customers c
```

```
  INNER JOIN orders o
    ON c.id_customers = o.id_customers
WHERE
  o.date_end IS NULL AND o.date_begin > '2018-02-10'
```

Этот запрос выведет информацию о заказах, находящихся в исполнении с датой начала заказа больше чем 10.02.2018 (таблица 2.9).

Таблица 2.9 — Результат выполнения запроса

customer	city	address	date_begin	date_end
Континент	Брянка	ул. Сува 95	15.02.2018	(null)
Квартал	Брянка	ул. Ленина 66	13.02.2018	(null)
Белый Квадрат	Брянка	ул. Абаканская 40	12.02.2018	(null)

Предикат ORDER BY позволяет определять порядок вывода строк в результирующем наборе. Синтаксис предиката:

```
ORDER BY
  <имя поля> [ASC | DESC]
  [{,  < имя поля > [ASC | DESC]}…]
```

Как показывает синтаксис, конструкция ORDER BY обязательно должна содержать, по крайней мере имя одного поля. Вместо фактического имени поля можно указывать его псевдоним. Если указываются несколько имен (псевдонимов) полей, они обязательно разделяются запятыми.

Для каждого поля, добавляемого в конструкцию ORDER BY, можно указывать, должны строки сортироваться в порядке возрастания (с помощью ключевого слова ASC) или в порядке убывания (с помощью ключевого слова DESC). Если ни одно из ключевых слов не указано, по умолчанию принимается ASC. Кроме того, когда перечисляются несколько полей, строки сортируются сначала по полю, указанным первым, далее — по полю, указанным вторым, и так далее.

Отсортируем таблицу customers сначала по городу (по алфавиту), а затем по уменьшению количества комнат:

```
SELECT
    c.name_customer, c.city
  , c.address,       o.date_begin
  , o.date_end,      o.payment
FROM
  customers c
  INNER JOIN orders o
    ON c.id_customers = o.id_customers
WHERE
  o.date_end IS NULL
  AND o.date_begin > '2018-02-10'
ORDER BY
  o.date_begin
```

2.1.4 Предикаты GROUP BY и HAVING

Оператор `SELECT` позволяет группировать данные — размещать данные в определенном логическом порядке в полях со значениями, которые повторяются. Группировка осуществляется с помощью ключевого слова `GROUP BY` с перечислением выбранных полей. Обычно с группировкой используются агрегирующие функции, используемые для обработки данных внутри каждой группы. Таких функций пять:

- `AVG()` — среднее значение;
- `SUM()` — сумма;
- `MIN()` — минимальное значение;
- `MAX()` — максимальное значение;
- `COUNT()` — количество значений.

Например, для подсчета количества договоров по заказчикам, выполняемых в текущее время (дата окончания договора пуста), применим такой запрос:

```
SELECT
    c.customer
  , c.city
  , c.address
  , COUNT(*) AS cnt
FROM
  customers c
```

```
  INNER JOIN orders o
    ON c.id_customers = o.id_customers
WHERE
  o.date_end IS NULL
GROUP BY
    c.customer
  , c.city
  , c.address
```

Результаты выполнения запроса приведены в таблице 2.10

Таблица 2.10 — Результат выполнения запроса

customer	**city**	**address**	**cnt**
Белый Квадрат	Брянка	ул. Абаканская 40	1
Гильдия	Брянка	ул. Ленина 15	2
Квартал	Брянка	ул. Ленина 66	2
Континент	Брянка	ул. Сува 95	1
Тетрис	Брянка	ул. Набережная 10	1

В функции `COUNT()` можно использовать ключевое слово `DISTINCT`, если необходимо, чтобы при подсчете количества значений дубликаты не учитывались, кроме того вместо выражения можно использовать знак «*», если необходимо, чтобы при подсчете учитывались все строки, независимо от того, содержат они значение `NULL` или нет. Если вместо символа «*» указано имя поля, в котором присутствуют значение `NULL`, строки, содержащие такие значения, при подсчете значений не учитываются.

Ключевое слово `HAVING` используется вместе с `GROUP BY` для того, чтобы указать, какие из групп должны быть представлены в результате запроса. Для `GROUP BY` ключевое слово `HAVING` играет ту же роль, что и `WHERE` для `ORDER BY`.

Например, выведем список заказчиков, для которых в текущий момент времени выполняется более одного договора:

```
SELECT
    c.customer, c.city
  , c.address
  , COUNT(*) AS cnt
```

```
FROM
  customers c
  INNER JOIN orders o
    ON c.id_customers = o.id_customers
WHERE
  o.date_end IS NULL
GROUP BY
    c.name_customer
  , c.city
  , c.address
HAVING
  cnt > 1
```

Результаты выполнения запроса приведены в таблице 2.11

Таблица 2.11 — Результат выполнения запроса

customer	city	address	cnt
Гильдия	Брянка	ул. Ленина 15	2
Квартал	Брянка	ул. Ленина 66	2

2.1.5 Предикат LIMIT

Предикат LIMIT используется для ограничения количества строк, возвращаемых оператором SELECT. LIMIT принимает один или два числовых аргумента, которые должны быть целочисленными константами. Когда указываются 2 аргумента, первый означает смещение в результирующем списке первой строки, а второй — максимальное количество возвращаемых строк. Смещение начальной строки равно 0 (не 1):

```
-- Вывод строк 6-15
SELECT * FROM table LIMIT 5, 10;
```

Чтобы вывести все строки, начиная с определенного смещения и до конца результирующего набора, можно использовать какое-нибудь большое число во втором аргументе. Этот оператор выводит все строки, начиная с 96-й и до последней:

```
SELECT * FROM table LIMIT 95, 18446744073709551515;
```

Если указан один аргумент, то он означает количество строк, которые следует вывести с начала результирующего набора:

```
-- Вывести первые 5 строк
SELECT * FROM table LIMIT 5;
```

Другими словами, `LIMIT n` эквивалентно `LIMIT 0, n`.

2.1.6 Функции в операторах SQL

Одним из элементов, используемых в выражениях SQL, являются функции. Функция выполняет определенную задачу, затем возвращает значение, являющееся результатом решения этой задачи. Многие функции требуют предоставления одного или более аргументов. Основные функции, которые используются в MySQL, приведены в таблицах 2.12 – 2.14.

Таблица 2.12 — Функции сравнения и преобразования данных

Функция	Описание
Функции сравнения	
GREATEST(), LEAST()	Сравнивают значения и возвращают наибольшее и наименьшее соответственно
COALESCE()	Возвращает первое значение в списке аргументов, которое не равняется NULL
ISNULL()	Возвращает 1, если выражение дает результат NULL, 0 - в остальных случаях
INTERVAL()	Сравнивает целое число с рядом целых чисел
STRCMP()	Сравнивает два строковых параметра
Функции управления потоком выполнением	
IF(), CASE()	Возвращают результат в зависимости от условия
IFNULL()	Возвращает <выражение1>, если оно не NULL, иначе возвращает <выражение2>
NULLIF()	Возвращает NULL, если <выражение1> = <выражение2>, иначе возвращает <выражение1>
Функции приведения	
CAST() CONVERT()	Превращают выражение к определенному типу данных

Таблица 2.13 — Функции работы с различными типами данных

Функция	Описание
Строковые функции	
ASCII(), ORD()	Возвращает числовое значение однобайтового и многобайтового символа соответственно
CHAR_LENGTH(), LENGTH()	Возвращает количество в строке символов и байтов соответственно
CONCAT()	Объединяет несколько строк
CONCAT_WS()	Позволяет при объединении строк вставлять между ними указанный разделитель
INSTR(), LOCATE()	Определяет, где именно в строке находится указанная подстрока
LCASE(), UCASE()	Изменяют регистр строки на нижней и верхней соответственно
LEFT(), RIGHT()	Возвращают указанное количество символов с левой и правой стороны строки соответственно
REPEAT()	Повторяет строку указанное количество раз
REVERSE()	Изменяет порядок символов в строке на противоположный
SUBSTRING()	Возвращает подстроку из середины строки
Числовые функции	
CEIL()	Возвращает наименьшее целое значение, не меньше указанного числа
FLOOR()	Возвращает наибольшее целое значение, не больше указанного значения
COT()	Котангенс числа
MOD()	Возвращает остаток от деления двух чисел
PI()	Возвращает значение числа π
POW()	Возводит число в указанную степень
ROUND(), TRANCATE()	Округляет числа
SQRT()	Возвращает квадратный корень из числа
Функции даты и времени	
ADDDATE()	Добавляет к дате указанный интервал времени
SUBDATE()	Отнимает от даты указанный интервал времени
EXTRACT()	Возвращает часть значения даты
CURDATE()	Возвращает текущую дату
CURTIME()	Возвращает текущее время
NOW()	Возвращает текущую дату и время
DATE()	Возвращает дату из значения даты времени
MONTH()	Возвращает число, соответствующее месяцу с даты

Продолжение таблицы 2.13

MONTHNAME()	Возвращает название месяца с даты
YEAR()	Возвращает год из даты
DATEDIFF()	Возвращает количество дней между датами
TIMEDIFF()	Возвращает разницу во времени между двумя значениями времени
DAY()	Возвращает из даты день месяца
DAYNAME()	Возвращает из даты название дня недели
DAYOFWEEK()	Возвращает из даты номер дня недели
DAYOFYEAR()	Возвращает из даты номер дня в году
SECOND()	Возвращает секунды из значения даты-времени
MINUTE()	Возвращает минуты из значения даты-времени
HOUR()	Возвращает часы из значения даты-времени
TIME()	Возвращает время из значения даты-времени

Таблица 2.14 — Функции выполнения системных операций

Функция	**Описание**
`ENCODE(), DECODE()`	Шифрует и дешифрует строчки соответственно
`CURRENT_USER()`	Возвращает имя пользователя в текущем сеансе
`DATABASE()`	Возвращает имя текущей базы данных

Описанные функции могут использоваться как отдельно, так и быть вложенными. Например, рассмотрим преобразования даты из формата «01.01.2010» к формату «01 January 2010»:

```
SELECT
    c.customer
  , c.city
  , c.address
  , CONCAT_WS(' ',
      DAY( o.date_begin),
      MONTHNAME( o.date_begin),
      YEAR( o.date_begin)) AS date
FROM
  customers c
  INNER JOIN orders o
    ON c.id_customer = o.id_customer
WHERE
  o.date_end IS NULL
```

Приведенный выше оператор выделяет из даты день, название месяца, год и объединяет их в одну строку, помещая между ними знак пробела (таблица 2.15).

Таблица 2.15 — Результат выполнения запроса

customer	city	address	date
Тетрис	Брянка	ул. Набережная 10	28 November 2017
Континент	Брянка	ул. Сува 95	15 February 2018
Квартал	Брянка	ул. Ленина 66	13 February 2018
Квартал	Брянка	ул. Ленина 66	8 February 2018
Гильдия	Брянка	ул. Ленина 15	2 February 2018
Гильдия	Брянка	ул. Ленина 15	5 February 2018
Белый Квадрат	Брянка	ул. Абаканская 40	12 February 2018
Тетрис	Брянка	ул. Набережная 10	28 November 2017

2.2 Добавление данных. Оператор INSERT

Оператор INSERT используется для непосредственного добавления данных в таблицу. Этот оператор предоставляет значительную степень гибкости для определения значений, как отдельных строк, так и для их множества. Оператор INSERT может иметь один из следующих синтаксисов:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
[INTO] имя_таблицы [{имя_поля, …}]
VALUES ({выражение | DEFAULT},…), (…),…
[ON DUPLICATE KEY UPDATE имя_поля=выражение, …]

INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
[INTO] имя_таблицы
SET имя_поля = {выражение | DEFAULT}, …
[ON DUPLICATE KEY UPDATE имя_поля=выражение, …]

INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
[INTO] имя_таблицы [(имя_поля, …)]
SELECT …
```

Оператор INSERT вставляет новые строки в существующую таблицу. Формы INSERT...VALUES и INSERT...SET этого оператора вставляют строки на основе явно указанных значений. Форма INSERT...SELECT позволяет быстро вставить множество строк в одну таблицу из другой (других).

Значение модификаторов команды INSERT следующее:

DELAYED — сервер помещает строку или строки, которые должны быть вставлены, в буфер, и клиент, который прислал этот оператор, может продолжать свою работу. Если таблица занята, сервер удерживает строки. Когда таблица освободится, он начнет их вставку, проверяя периодически, нет ли новых запросов на чтение этой таблицы. Если они есть, обслуживание очереди отложенных строк для вставки приостанавливается до тех пор, пока таблица не освободится снова;

LOW_PRIORITY — выполнение вставки откладывается до тех пор, пока все остальные клиенты не завершат чтение таблицы;

IGNORE — любая строка, в которой дублируется значение полей уникального индекса или первичного ключа, игнорируется и не вставляется. Если вы не указываете IGNORE, операция вставки прерывается при обнаружении дублированных строк.

Если вы используете конструкцию ON DUPLICATE KEY UPDATE, и вставляется строка с дублированным значением ключа уникального индекса или первичного ключа, то выполняется операция UPDATE старой строки. При этом опция DELAYED игнорируется.

Запишем оператор INSERT для занесения данных в таблицу customers данных о новом заказчике:

```
INSERT INTO customers(customer, city, address)
VALUES (1, 'Сантех', 'Алчевск', 'ул. Советская 55');
```

Этот же запрос можно записать так:

```
INSERT INTO customers
SET
    customer = 'Сантех', city = 'Алчевск''
  , address = 'ул. Советская 55';
```

Если существует необходимость переноса данных из одной таблицы в другую, это можно сделать с помощью следующего оператора INSERT:

```
INSERT INTO new_customers(customer, city, address)
SELECT
  customer, city, address
FROM
  customer
WHERE
  customer = 'Квартал';
```

Необязательно перечислять названия полей таблицы, в которую вставляются данные, но тогда необходимо проверить, что данные заносятся в правильном порядке.

2.3 Обновление данных. Операторы UPDATE и REPLACE

Оператор UPDATE обновляет поля существующих строк таблицы новыми значениями. Он позволяет обновлять данные как в одной, так и в нескольких таблицах одновременно.

Однотабличный синтаксис:

```
UPDATE [LOW_PRIORITY] [IGNORE] имя_таблицы
SET имя_поля1 = выражение1 [, имя_поля2 = выражение2 …]
[WHERE определение_WHERE]
{ORDER BY …]
[LIMIT количество_строк]
```

Например, обновим адрес для заказчика с наименованием «Квадрат»:

```
UPDATE customers c
SET
  c.address = 'ул. Парковая 8'
WHERE
  c.customer = 'Квадрат';
```

Многотабличный синтаксис UPDATE:

```
UPDATE [LOW_PRIORITY] [IGNORE]
  имя_таблицы [, имя_таблицы …]
SET
  имя_поля1 = выражение1 [, имя_поля2 = выражение2 …]
[WHERE определение_WHERE]
```

Конструкция SET перечисляет обновляемые поля и значение, которые им присваиваются. Если указана конструкция WHERE, она задает, какие строки должны быть обновлены. В противном случае обновляются все строки таблицы. Если указана конструкция ORDER BY, строки будут обновлены в заданном порядке. Конструкция LIMIT накладывает ограничения на количество строк, которые обновляются. Оператор UPDATE завершает работу, как только найдет заданное количество строк, удовлетворяющих условию WHERE, независимо от того, были ли они действительно обновленные.

В многотабличных операторах UPDATE нельзя применять ORDER BY и LIMIT.

Как результат оператор UPDATE возвращает количество строк, которые были обновлены.

Оператор REPLACE работает так же, как INSERT, за исключением того, что если строка с тем же значением первичного ключа в таблице существует, то старая строка удаляется перед вставкой новой. Надо отметить, что использование REPLACE не имеет смысла, если таблица не имеет первичных ключей. В этом случае оператор полностью эквивалентен INSERT. Синтаксис:

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] имя_таблицы [{имя_поля, …}]
VALUES ({выражение | DEFAULT},…), (…),…

REPLACE [LOW_PRIORITY | DELAYED]
[INTO] имя_таблицы
SET имя_поля = {выражение | DEFAULT}, …

REPLACE [LOW_PRIORITY | DELAYED] [IGNORE]
[INTO] имя_таблицы [(имя_поля, …)]
SELECT …
```

2.4 Удаление данных. Операторы DELETE и TRUNCATE

Оператор DELETE — это основной оператор, используемый для удаления данных из таблицы. Синтаксис оператора DELETE:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
FROM
  имя_таблицы
[WHERE определение_WHERE]
{ORDER BY имя_поля [ASC | DESC] [{,
       <имя_поля> [ASC | DESC]} … ]]
[LIMIT количество_строк]
```

Оператор удаления данных можно использовать для удаления из базы данных заданного заказчика:

```
DELETE FROM customers
WHERE
  customer = 'Квадрат'
```

Оператор TRUNCATE полностью очищает таблицу. Синтаксис:

```
TRUNCATE TABLE имя_таблицы
```

Логично, что это эквивалент оператора DELETE, который удаляет все строки, но в некоторых случаях имеются определенные различия практического плана:

– оператор TRUNCATE небезопасен в отношении транзакций (отмена удаления невозможна);

– оператор TRUNCATE, в отличие от DELETE, начинает нумерацию AUTO_INCREMENT заново;

– обычно TRUNCATE работает быстрее DELETE.

2.5 Создание запросов в среде dbForge Studio for MySQL

Для создания запроса в среде dbForge Studio for MySQL существует несколько способов:

– с помощью кнопок «Новый запрос» и «Новый SQL» на панели инструментов «Стандартная»;

– с помощью команд главного меню «Файл → Создать → Запрос» и «Файл → Создать → SQL»;

– с помощью команд «Новый запрос» и «Новый SQL» контекстного меню базы данных в окне «Проводника баз данных».

Команда «Новый SQL» открывает пустое окно, в котором необходимо записывать SQL-операторы. Команда «Новый запрос» позволяет создавать запросы в режиме конструктора.

На рисунке 2.1 приведено окно конструктора запросов. Оно состоит из трех частей. В первой части отображаются таблицы, используемые в запросе. Их можно перетащить мышкой из «Проводника баз данных». Вторая часть окна имеет два режима. В режиме «Дизайн» во второй части окна отображаются поля, которые будет выводить запрос (есть возможность задать псевдоним поля, тип сортировки и условия отбора строк). На вкладке «Группировка» можно выбрать поля, по которым будут группироваться результаты запроса. В режиме «Текст» отображается текст SQL-запроса, автоматически формируется на основе данных, введенных в режиме «Дизайн». Пользователь имеет возможность корректировать этот текст.

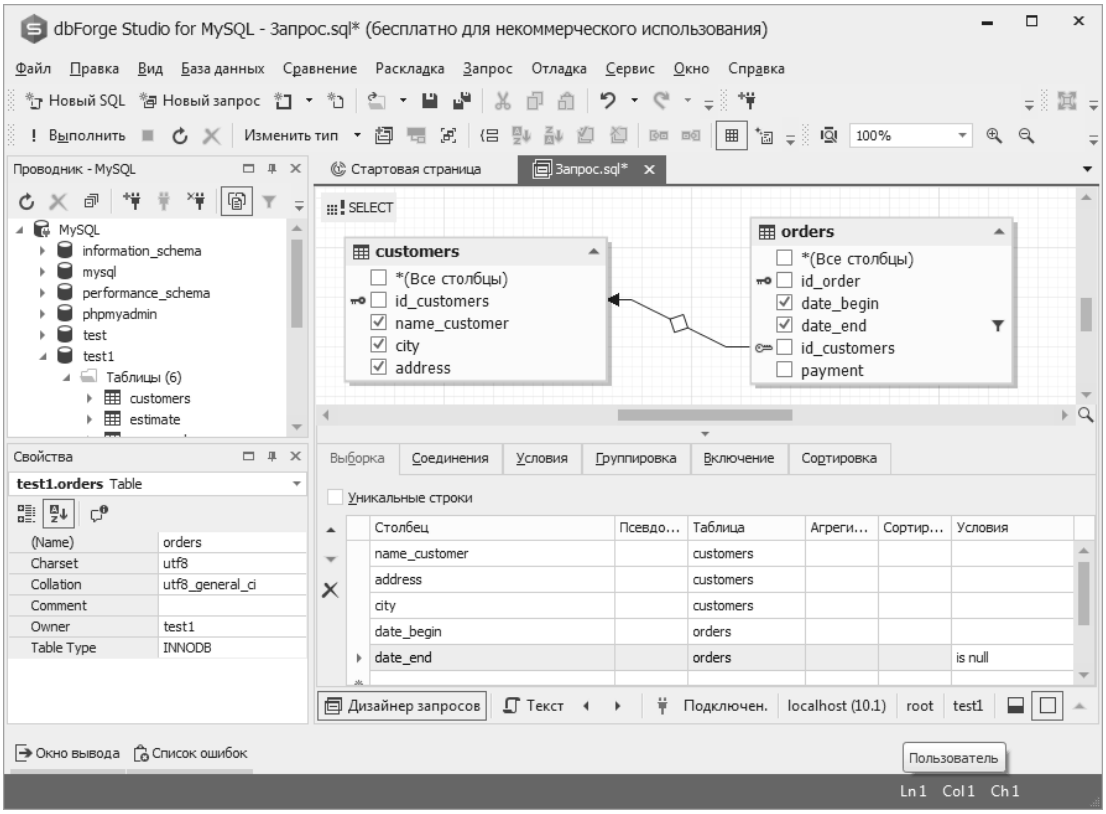

Рисунок 2.1 — Окно конструктора запроса

Для выполнения запроса необходимо нажать кнопку «Выполнить» на панели инструментов «Запрос» или клавишу F5. Результаты запроса отображаются в нижней части окна запроса.

2.6 Основные понятия о триггерах

Триггер — это процедура, вызываемая событием. Но, в отличие от процедур, триггер привязан не к базе данных, а к таблице, и вызывается событием только в той таблице, к которой он привязан.

Для создания триггера применяется команда SQL `CREATE TRIGGER`. Синтаксис:

```
CREATE [DEFINER = CURRENT_USER]
TRIGGER `[ имя триггера]`
[BEFORE | AFTER ] [ INSERT | UPDATE | DELETE]
ON `[ имя таблицы ]`
FOR EACH ROW
BEGIN
  [ Тело триггера ]
END;
```

Параметр `DEFINER` определяет пользователя, создающего триггер. В MySQL для осуществления операций по созданию или удалению триггеров требуется наличие у пользователя привилегии `SUPER`.

Триггер можно создавать на 3 события: `INSERT`, `UPDATE`, `DELETE`. Для каждого из этих событий имеется возможность управлять моментом срабатывания триггера: «до» события (`BEFORE`) и «после» события (`AFTER`). Таким образом, получаем 6 типов триггеров.

Заметим, что событие `INSERT` происходит, когда новая строка добавляется в таблицу. Например, из команды `INSERT`, `LOAD DATA` или `REPLACE`. `UPDATE` срабатывает, когда строка изменяется через инструкцию `UPDATE`, а `DELETE` — когда строка удаляется из таблицы, например, через инструкции `DELETE` и `REPLACE`. Однако, инструкции `DROP TABLE` и `TRUNCATE` не активизируют триггер, т.к. они не используют `DELETE`!

В теле триггера задаются инструкции, выполняемые при активации триггера.

В триггерах MySQL используются псевдонимы OLD и NEW. Они позволяют обращаться к данным в разном состоянии. OLD.col_name (col_name — имя поля) обращается к полю существующей строки перед тем, как он будет изменена или удалена. NEW.col_name обращается к полю добавляемой (вставляемой) строки, или к существующей строке после того, как она будет модифицирована.

Создадим систему триггеров для таблицы workers базы данных DB, позволяющих вести историю изменений данных. То есть при удалении или изменении элемента из таблицы будет создаваться его резервная копия в специальной таблице. Определение: workers — таблица, над которой проводят операции, id_worker — уникальное поле (` workers `.` id_worker `), arch_mytable — название таблицы-архива.

Создаем пустой дубликат таблицы:

```
CREATE TABLE arch_workers
SELECT * FROM  workers LIMIT 0;
```

Добавляем в таблицу новые поля: trigdate (дата операции); trigact (тип операции «del» или «upd»):

```
ALTER TABLE arch_workers
    ADD COLUMN `trigdate` DATETIME  NULL
  , ADD COLUMN `trigact`  CHAR(3)   NULL;
```

Создаем триггер для таблицы workers для события DELETE:

```
CREATE TRIGGER archiver_workers_del
  BEFORE DELETE ON workers FOR EACH ROW
INSERT INTO arch_workers
SELECT
  *, CURRENT_TIMESTAMP AS trigdate, del AS trigact
FROM
  workers
WHERE
```

```
  id_worker = OLD.id_worker;
```

Создаем триггер для таблицы workers для события UPDATE:

```
CREATE TRIGGER archiver_workers_upd
  BEFORE UPDATE ON workers FOR EACH ROW
INSERT INTO arch_workers
SELECT
  *, CURRENT_TIMESTAMP AS trigdate, upd AS trigact
FROM
  workers
WHERE
  id_worker = OLD.id_worker;
```

В MySQL есть некоторые ограничения на инструкции, используемые в триггерах:

– триггер не может использовать инструкцию CALL, чтобы вызвать хранимые процедуры, возвращающие данные пользователю или применяют динамический SQL. Процедурам разрешается вернуть данные в триггер через параметры OUT или INOUT;

– триггер не может использовать инструкции, которые явно или неявно начинают или заканчивают транзакцию, типа START TRANSACTION, COMMIT или ROLLBACK;

MySQL обрабатывает ошибки при выполнении триггеров следующим образом:

– если проблемы с триггером BEFORE, операции на соответствующей строке просто не выполняются;

– триггер BEFORE активизируется попыткой вставить или изменить строку независимо от того, была ли попытка удачной;

– триггер AFTER выполняется только, если триггер BEFORE и операция со строкой выполняются успешно;

– ошибка в триггере BEFORE или AFTER вызывают сбой всей инструкции, вызвавшей триггер;

– для транзакционных таблиц сбой инструкции должен вызвать откат всех изменений, вызванных инструкцией.

Для удаления существующего триггера используется команда DROP TRIGGER:

```
DROP TRIGGER trigger_name
```

Изменение триггера осуществить нельзя, но можно использовать конструкцию DROP TRIGGER IF EXISTS `trigger` для удаления триггера, а затем создать новой с таким же именем.

2.7 Хранимые процедуры и функции в MySQL

Хранимая процедура — это набор SQL-операторов, хранящихся на сервере. После этого пользователям не нужно будет повторно задавать те же отдельные операторы, вместо этого они будут обращаться к хранимой процедуре.

Хранимые процедуры помогают повысить эффективность, так как при их использовании объем пересылаемой информации между сервером и клиентом, существенно уменьшается. Хранимые процедуры также позволяют создавать библиотеки функций на сервере баз данных.

Хранимые процедуры и функции представляют собой подпрограммы, создаваемые с помощью операторов CREATE PROCEDURE и CREATE FUNCTION. Подпрограмма является или процедурой, или функцией. Процедура вызывается с помощью оператора CALL, и может только передавать значения обратно с использованием переменных. Функции могут возвращать скалярное значение и вызываться из оператора так же, как и любые другие функции (через указания имени функции). Хранимые процедуры могут вызвать другие хранимые процедуры.

Синтаксис хранимой процедуры:

```
CREATE PROCEDURE <имя_процедуры>
([ параметр [, …])
[BEGIN]
  <SQL-операторы>
[END]
```

Параметры задаются в формате:

```
[IN | OUT | INOUT] <имя параметра> < тип>.
```

Синтаксис функции:

```
CREATE FUNCTION <имя_функции>  ([ параметр [, …])
[RETURNS тип]
[BEGIN]
  <SQL-операторы>
[END]
```

Конструкция RETURN может быть определена только для FUNCTION. Она используется для указания типа результата функции, при этом в теле функции должен присутствовать оператор RETURN значения.

Список аргументов в круглых скобках должен присутствовать всегда. Если аргументы отсутствуют, используйте пустой список аргументов (). Каждый аргумент по умолчанию является аргументом IN. Для указания другого, перед названием аргумента укажите ключевое слово OUT или INOUT. Значение IN, OUT, INOUT доступны только для PROCEDURE.

Ниже приведен пример хранимой процедуры, которая выбирает информацию о заказчиках, наименование города расположения которых равно параметру par1:

```
CREATE PROCEDURE proc1(IN par1 varchar(30))
BEGIN
  SELECT * FROM customers WHERE city = par1;
END
```

Вызов процедуры осуществляется строкой:

```
CALL proc1('Алчевск');
```

Следующая функция вычисляет значение дискриминанта квадратного уравнения на основе коэффициентов, которые передаются в качестве параметров функции:

```
CREATE FUNCTION Det(a int, b int, c int) RETURNS float
BEGIN
  RETURN (b * b – 4 * a * c);
END
```

Для удаления хранимых процедур или функций используется следующая команда:

```
DROP (PROCEDURE | FUNCTION) <имя процедуры (функции)>
```

2.8 Порядок выполнения работы

1. Для созданной в лабораторной работе №1 базы данных создайте запросы, триггер и хранимую процедуру согласно индивидуальному заданию. Если запрос не выдает результат, скорректируйте данные в таблицах базы данных.

2. Распечатайте SQL-тексты запросов, триггера и процедуры, а также результаты их выполнения.

2.9 Варианты заданий

Вариант 1. База данных банка

1. Создать запрос, который выбирает сотрудников в возрасте от 30 до 50 лет (КодСотрудника, ФИО, Возраст, НазДолжности). Список отсортировать сначала в соответствии с должностью (по алфавиту), а затем в соответствии с уменьшением возраста.

2. Создать запрос, который выводит общую сумму невозвращенных вкладов по типам (НазТипа, Ставка, Невозвращенная сумма).

3. Создать запрос, который выводит количество принятых вкладов по годам (Год, КоличествоВкладов). Список отсортировать по уменьшению количества вкладов.

4. Создать триггер, который при удалении информации о вкладе в таблице «Вклады» удаляет запись о вкладчике из таблицы «Вкладчики», если у него больше нет других вкладов.

5. Используя текст запроса из п.1, создать хранимую процедуру, которая принимает два значения возраста сотрудников и выводит список сотрудников, чей возраст находится между значениями этих параметров.

Вариант 2. База данных больницы

1. Создать запрос, который выводит информацию о всех пациентах с именем, начинающимся на буквы «П» или «К». Список отсортировать по алфавиту (ФИО, Возраст, Адрес, Телефон).

2. Создать запрос, который выводит суммарные расходы на лекарства за предыдущий календарный год (НазЛекарства, СуммарныеРасходы). Список отсортировать по сокращению расходов.

3. Создать запрос, выводящий список из трех болезней, которыми чаще всего болеют пациенты зимой (НазБолезни, Месяц, Количество пациентов).

4. Создать триггер, который при добавлении информации о новых лекарствах переводит цену в евро.

5. Используя текст запроса из п.1, создать хранимую процедуру, которая в качестве параметра принимает букву и выводит список пациентов, имя которых начинается на эту букву.

Вариант 3. База данных отеля

1. Создать запрос, который рассчитывает выручку отеля по месяцам (Месяц, Выручка). Список отсортировать по уменьшению выручки.

2. Создать запрос, выводящий список из трех клиентов, которые потратили наибольшее количество денег за предыдущий месяц (ФИО, Сумма). Список отсортировать па алфавиту.

3. Создать запрос, который выводит виды услуг ценой от 200 до 500 руб. (НазУслуги, Описание, Цена).

4. Создать триггер, который при удалении записи о клиенте записывает информацию о нем в таблицу «Архив» (ФИО, Паспорт, ДатаУдаления, КтоУдалил).

5. Используя текст запроса из п.3, создать хранимую процедуру, которая принимает два значения цены в качестве параметров и выводит все услуги, цены на которые находятся в интервале между введенными значениями.

Вариант 4. База данных аэропорта

1. Создать запрос, выводящий пункты назначения, в которые за последние две недели не было осуществлено ни одного рейса (КодГорода, НазГорода). Список отсортировать по наименованию городов (по алфавиту).

2. Создать запрос, который выводит список самолетов вместимостью от 150 людей (Марка, Вместимость, ДатаВыпуска). Список отсортировать в порядке уменьшения даты выпуска.

3. Создать запрос, который выводит список из пяти пассажиров, которые чаще всех летают летом.

4. Создать триггер, который при вставке записи в таблицу «ПроданныеБилеты» заносит запись в таблицу «ЖурналУчета» (ДатаПродажи, Пользователь, ПунктНазначения, Цена, Место).

5. Используя текст запроса из п.3, создать хранимую процедуру, которая принимает в качестве параметров две даты и выводит список из 5 пассажиров, которые чаще всех летают в промежуток времени между введенными значениями дат.

Вариант 5. База данных проката дисков

1. Создать запрос, который выводит список сотрудников, отсортированный по алфавиту (все поля).

2. Создать запрос, выводящий список клиентов, которые имеют диски на руках (ФИО, Телефон, КоличествоДисков). Список отсортировать по уменьшению количества дисков.

3. Создать запрос, выводящий информацию о 2 жанрах, которые пользуются наибольшей популярностью у мужчин (НазЖанру, КоличествоЗапросов).

4. Создать триггер, который при заполнении даты возвращения диска заносит информацию о прокате в таблицу «Возвращения» (ФИОклиента, НазваниеФильма, Дата взятия, ДатаВозвращения, СтоимостьПроката).

5. Используя текст запроса из п.3, создать хранимую процедуру, которая принимает в качестве параметра пол и выводит два жанра, которые пользуются наибольшей популярностью у клиентов соответствующего пола.

Вариант 6. Базы данных библиотеки

1. Создать запрос, выводящий три жанра, которые пользовались наибольшей популярностью за прошлый месяц (НазЖанра, КоличествоВыдач). Список отсортировать по уменьшению количества выдач.

2. Создать запрос, который выводит список читателей в возрасте от 30 лет (ФИО, Возраст, Адрес, Телефон). Список отсортировать по увеличению возраста.

3. Создать запрос, выводящий список книг, которыми пользовались больше месяца (НазКниги, ФИОЧитателя, ДлительностьЧтения).

4. Создать триггер, который в случае добавления записи в таблицу «Читатели» копирует информацию о читателе в таблицу «Мужчины» или «Женщины» в зависимости от пола читателя. Таблицы должны содержать поля: КодЧитателя, ФИО, Возраст.

5. Используя текст запроса из п.2, создать хранимую процедуру, которая принимает в качестве параметра возраст и выводит список читателей, чей возраст превышает введенное значение.

Вариант 7. База данных радиостанции

1. Создать запрос, выводящий список сотрудников с фамилиями, которые начинают на буквы от «О» до «Я» (ФИО, Дата рождения, Пол). Список отсортировать по алфавиту.

2. Создать запрос, выводящий информацию об исполнителях, которые за последние полгода не записали ни одной песни (Название (ФИО), Описание).

3. Создать запрос, который выводит пять записей, которые чаще всего ставят в утреннее (до 12-00) время (НазЗаписи, Исполнитель, Жанр, КоличествоРотаций). Список отсортировать по уменьшению количества ротаций.

4. Создать триггер, который в случае удаления исполнителя удаляет все связанные с ним записи в таблице записи.

5. Используя текст запроса из п.3, создать хранимую процедуру, которая принимает в качестве параметров два значения времени и выводит 5 записей, которые чаще всего ставят в промежуток времени между заданными значениями.

Вариант 8. База данных таксопарка

1. Создать запрос, выводящий список водителей, которые родились весной (ФИО, Дата рождения, Пол). Список отсортировать по дате рождения.

2. Создать запрос, который рассчитывает выручку таксопарка за предыдущий календарный месяц (Выручка).

3. Создать запрос, выводящий список автомобилей, которые не использовались в течение последних десяти дней.

4. Создать триггер, который в случае изменения размера тарифа в таблице «Тарифы» заносит информацию об изменениях в таблицу «УчетТарифов» (Дата изменения, Пользователь, СтарыйТариф, НовыйТариф, ПроцентИзменения).

5. Используя текст запроса из п.3, создать хранимую процедуру, которая принимает в качестве параметра целое число и выводит список автомобилей, которые не использовались в течение соответствующего количества дней.

Вариант 9. База данных туристического агентства

1. Создать запрос, выводящий клиентов, которые не пользовались дополнительными услугами (ФИО, ДатаПоселения, ДатаВыезда).

2. Создать запрос, выводящий информацию о двух гостиницах, в которых за лето останавливалось наибольшее количество клиентов (Гостиница, Город, КоличествоКлиентов).

3. Создать запрос, который выводит список путевок стоимостью больше 10000 руб. (КодПутевки, Гостиница, Клиент, ДатаПоселения, Стоимость). Список отсортировать сначала по гостиницам (по алфавиту), а потом по уменьшению стоимости путевки.

4. Создать триггер, который в случае удаления гостиницы переносит информацию о ней в таблицу «Архив» (КодГостиницы, НазваниеГостиницы, Город, Адрес, КоличествоЗвезд, ДатаУдаления, КтоУдалил).

5. Используя текст запроса из п.3, создать хранимую процедуру, которая принимает в качестве параметра стоимость путевки и выводит список путевок, стоимость которых превышает введенное значение.

Вариант 10. База данных страховой компании

1. Создать запрос, который подсчитывает среднюю сумму выплаты по каждой группе клиентов (НазГруппы, СредняяСумма). Список отсортировать в порядке уменьшения суммы.

2. Создать запрос, который выводит данные об оформленных полисах сроком действия от одного года (КодПолиса, ДатаОформления,

СрокДействия, ФИОклиента, НазФилиалу). Список отсортировать в порядке уменьшения срока действия.

3. Создать запрос, выводящий список филиалов, которые в проходящем месяце не оформили ни одного полиса (НазФилиала, Адрес, Телефон).

4. Создать триггер, который в случае добавления новой записи в таблицу «Полисы» значения, которое вводится в поле «СуммаВыплаты» увеличивает на 30%.

5. Используя текст запроса из п.2, создать хранимую процедуру, которая принимает в качестве параметра количество лет и выводит данные об оформленных полисах, срок действия которых соответствует введенному значению.

Вариант 11. База данных сервисного центра

1. Создать запрос, выводящий список из трех неисправностей, которые случались чаще всего за прошлый месяц (НазНеисправности, Количество). Список отсортировать в порядке уменьшения количества случаев.

2. Создать запрос, выводящий список запчастей, цена которых варьируется от 250 до 500 руб. Список отсортировать в соответствии с ростом цены.

3. Создать запрос, который подсчитывает стоимость уже проведенных ремонтов (тех, для которых заполнена дата возвращения). Вывести поля: ДатаЗаказа, Магазин, Неисправность, Стоимость.

4. Создать триггер, который в случае добавления новой записи в таблицу «Заказа» будет проверять, если этот вид неисправности уже дсять раз встречается в этой таблице, то будет заносить информацию о нем в таблицу «Нехватка» (КодВида, НазВида, Описание, ДатаЗаноса).

5. Используя текст запроса из п.3, создать хранимую процедуру, которая в качестве параметров принимает две цены и выводит список запчастей, цена на которые варьируется между введенными значениями.

Вариант 12. База данных транспортной компании

1. Создать запрос, выводящий четыре вида грузов, которые чаще всего перевозили в течение последнего полугодия (НазГруза, Количество рейсов). Список отсортировать по алфавиту.

2. Создать запрос, который подсчитывает среднюю стоимость перевозок по месяцам (Месяц, СредняяСтоимость).

3. Создать запрос, который выводит список водителей, старше 50 лет (ФИО, ДатаРождения, Возраст). Список отсортировать по увеличению возраста.

4. Создать триггер, который записывает информацию о выполненных рейсах по перевозке грузов весом больше 100 кг в таблицу «ГабаритныеГрузы» (Дата, НазГруза, Вес, СтоимостьДоставки).

5. Используя текст запроса из п.2, создать хранимую процедуру, которая принимает в качестве параметра номер месяца и выводит среднюю стоимость перевозок за указанный месяц.

Вариант 13. База данных деканата

1. Создать запрос, выводящий список преподавателей, которые ведут больше одного предмета (ФИО, НазКафедры, Количество). Список отсортировать сначала по наименованию кафедры, а потом по ФИО преподавателя (по алфавиту).

2. Создать запрос, выводящий список, который отображает средний балл по каждому предмету за предыдущий календарный год.

3. Создать запрос, выводящий список студентов, которые не сдали ни одного предмета (НомерСтудБилета, ФИО).

4. Создать триггер, который заносит данные о предметах, которые ведут больше чем один преподаватель, в таблицу «Совместители» (КодПредмета, Название, ФИОПреподавателя).

5. Используя текст запроса из п.2, создать хранимую процедуру, которая принимает в качестве параметра название предмета и выводит средний балл этого предмета за предыдущий календарный год.

Вариант 14. База данных строительной компании

1. Создать запрос, который подсчитывает средние расходы на материалы по каждому заказчику (ФИО, СредниеРасходы). Список отсортировать по алфавиту.

2. Создать запрос, который выводит список незаконченных работ (ФИОЗаказчика, ВидРаботы, ДатаНачала). Список отсортировать в соответствии с увеличением даты начала работ.

3. Создать запрос, который выводит перечень материалов с ценой от 250 до 520 руб.

4. Создать триггер, который будет отслеживать историю регистрации сотрудников — в случае изменения адреса регистрации записывать информацию в таблицу «Регистрация» (ФИО, ДатаРождения, НовыйАдрес, ДатаИзменения).

5. Используя текст запроса из п.3, создать хранимую процедуру, которая принимает в качестве параметров две цены и выводит список материалов, цена которых варьируется в заданном диапазоне.

Вариант 15. База данных риэлтерской фирмы

1. Создать запрос, который выводит список сотрудников, серия паспорта которых «ЕК» (ФИО, ДатаРождения, Паспорт). Список отсортировать по алфавиту.

2. Создать запрос, который рассчитывает максимальное количество комнат среди квартир каждого вида (НазВида, КоличествоКомнат).

3. Создать запрос, выводящий информацию о договорах, которые были заключены весной прошлого года (Дата, ФИОклиента, АдресКвартиры, НазУслуги, СуммаДоговора). Список отсортировать в соответствии с уменьшением суммы договора.

4. Создать триггер, который в случае удаления информации о квартире перемещает записи о договорах, связанные с ней, в таблицу «Архив» (КодКвартиры, Адрес, ВидУслуги, ДатаУдаления).

5. Используя текст запроса из п.1, создать хранимую процедуру, которая в качестве параметра принимает идентификатор серии паспорта и выводит список сотрудников, серия паспорта которых совпадает с введенным значением.

Вариант 16. База данных рекламного агентства

1. Создать запрос, который выводит список неуплаченных заказов за последний месяц (ДатаЗаказа, НазваниеЗаказчика, ДатаВыполнения). Список отсортировать в соответствии с уменьшением даты заказа.

2. Создать запрос, выводящий список из трех сотрудников, которые предоставили наибольшее количество услуг по заказам (ФИО, Количество услуг).

3. Создать запрос, который отбирает услуги стоимостью от 400 до 1000 руб. (НазУслуги, Описание, Стоимость). Список отсортировать в порядке уменьшения стоимости.

4. Создать триггер, который при добавлении нового сотрудника копирует данные о нем в таблицу «МолодыеСотрудники» (ФИО, Возраст, Адрес, Пол), если его возраст не превышает 35 лет.

5. Используя текст запроса из п.3, создать хранимую процедуру, которая принимает в качестве параметров две цены и выводит список услуг, стоимость которых попадает в заданный интервал.

Вариант 17. База данных компьютерной фирмы

1. Создать запрос, который выводит суммарное количество заказанных комплектующих каждого вида (НазВида, Количество). Список отсортировать по уменьшению количества.

2. Создать запрос, который выводит список комплектующих со сроком гарантии больше одного года (Название, НазВида, Марка, ДатаВыпуска, СрокГарантии). Список отсортировать сначала по виду (по алфавиту), а потом в соответствии с увеличением срока гарантии.

3. Создать запрос, который выводит список из трех заказчиков, которые заказали на наименьшую сумму за последние две недели (Фамилия, Адрес, Сумма).

4. Создать триггер, который в случае установления отметки «Оплачен» копирует данные о выполненных заказах в таблицу «ВыполненныеЗаказы» (ДатаВыполнения, ФИОзаказчика).

5. Используя текст запроса из п.3, создать хранимую процедуру, которая в качестве параметров принимает две даты и выводит список из трех заказчиков, которые заказали на наименьшую сумму в интервале времени между введенными значениями.

Вариант 18. База данных ГАИ

1. Создать запрос, который подсчитывает количество автомобилей каждой марки (НазваниеМарки, Количество). Список отсортировать в соответствии с уменьшением количества.

2. Создать запрос, который выводит список водителей с просроченными водительскими правами (ФИО, НомерВодителяПрава, ДатаВыдачи, СрокДействия).

3. Создать запрос, который формирует зарплатную ведомость за прошлый календарный месяц (ФИО, Оклад, Надбавки, Всего).

4. Создать триггер, который в случае добавления нового автомобиля записывает в таблицу «ЧерныеАвто» (КодАвто, НазваниеМарки, ДатаВыпуска, РегНомер) информацию о нем, если цвет нового авто черный.

5. Используя текст запроса из п.3, создать хранимую процедуру, которая принимает в качестве параметра дату и выводит список водителей, срок действия водительского удостоверения которых заканчивается к указанной дате.

Вариант 19. База данных кинотеатра

1. Создать запрос, выводящий список фильмов, на которые не было продано ни одного билета (НазваниеФильма, НазваниеЖанра, Длительность). Список отсортировать в соответствии с увеличением длительности.

2. Создать запрос, выбирающий пять стран, которые выпустили наибольшее количество фильмов за последний год (НазваниеСтраны, КоличествоФильмов). Список отсортировать по алфавиту.

3. Создать запрос, который подсчитывает выручку от продажи билетов за прошлый календарный месяц.

4. Создать триггер, который в случае удаления фильма заносит информацию о нем в таблицу «Архив» (НазваниеФильма, НазЖанру, ДатаУдаления, Пользователь)

5. Используя текст запроса из п.3, создать хранимую процедуру, которая принимает в качестве параметра номер месяца и выводит суммарную выручку от продажи билетов за указанный месяц текущего года.

Вариант 20. База данных автосалона

1. Создать запрос, который подсчитывает среднюю цену автомобилей каждого типа кузова (ТипКузова, СредняяЦена). Список отсортировать по убыванию цены.

2. Создать запрос, выводящий список автомобилей, которые были проданы осенью (ДатаПродажи, Марка, Цвет, Цена, ФИОзаказчика, ФИОсотрудника). Список отсортировать сначала по маркам (по алфавиту), а потом по росту цены.

3. Создать запрос, который выводит список клиентов, которые купили более одного автомобиля (Фамилия, Адрес, Телефон, Количество).

4. Создать триггер, который в случае изменения цены на авто заносит информацию об изменении в таблицу «УчетЦен» (ДатаИзменения, КодАвто, Марка, СтараяЦена, НоваяЦена, ПроцентИзменения).

5. Используя текст запроса из п.1, создать хранимую процедуру, которая принимает в качестве параметра название типа кузова и выводит среднюю цену автомобилей с указанным типом кузова.

2.10 Требования к отчету

Отчет оформляется на стандартных листах белой бумаги формата А4 (210х297 мм). Текст пишется с одной стороны листа чернилами синего или черного цвета или печатается на принтере.

Бланк отчета готовится во время домашней подготовки к работе и должен содержать:

– данные о студенте: фамилия и инициалы, шифр группы;

– тему и цель работы;

– дату выполнения работы;

– набор данных, необходимых для выполнения работы: задание к работе, SQL- текст запросов, триггеров и сохраненных процедур, результаты выполнения, краткие теоретические сведения об их создании.

После выполнения работы к отчету подшиваются: распечатка текстов запросов, триггеров и сохраненных процедур, а также результаты их выполнения. Отчет должен быть выполнен аккуратно. Неаккуратно выполненные отчеты, а также ксерокопии чужих отчетов или их фрагментов к защите не допускаются.

2.11 Контрольные вопросы

1. Для чего предназначен оператор SELECT?

2. Общий синтаксис оператора SELECT.

3. Что означает ключевое слово DISTINCT в операторе SELECT?

4. Какое ключевое слово используется для сортировки строк в запросе?

5. Как интерпретируется в различных частях оператора SELECT символ «*»?

6. Для чего используется конструкция INNER JOIN, LEFT JOIN, RIGHT JOIN, их синтаксис.

7. Какие виды операций в операторе SELECT вы знаете?

8. Назначение предиката HAVING.

9. Перечислите агрегирующие функции SQL и дайте их краткое описание.

10. Для чего используется предикат LIMIT. Его синтаксис.

11. Какие вы знаете функции преобразования данных? Функции управления потоком выполнения?

12. Назначение оператора INSERT. Общий синтаксис.

13. Сравните операторы обновления данных UPDATE и REPLACE.

14. Какая особенность применения многотабличного оператора UPDATE?

15. Чем оператор DELETE отличается от оператора TRUNCATE?

16. Для чего используются триггеры?

17. Какие вы знаете типы триггеров? Когда они срабатывают?

18. Что такое хранимая процедура?

19. Какой оператор используется для вызова хранимой процедуры и каков его синтаксис?

20. Что такое хранимая функция, каково ее отличие от хранимой процедуры?

3 ЛАБОРАТОРНАЯ РАБОТА № 3

Тема: программирование баз данных в Visual C#.NET.

Цель работы: закрепить теоретический материал по программирование баз данных в Visual C#.NET.

3.1 Основные понятия технологии ADO.NET

Технология ADO.NET — следующий шаг Microsoft в развитии технологии Active Data Objects. Эта модель доступа к данным создана специально для использования в Windows и Web-приложениях.

В ADO.NET используется модель работы пользователя,– отключенного от источника данных. Приложение подключается к базе данных только на небольшой промежуток времени. Соединение устанавливается только тогда, когда клиент на удаленном компьютере запрашивает данные на сервере. После того, как сервер подготовил необходимый набор данных, сформировал и отправил их клиенту в виде WEB-страницы, связь приложения с сервером сразу же обрывается, и клиент просматривает полученную информацию уже без связи с сервером. При работе в сети Интернет нет необходимости поддерживать постоянную «жизнеспособность» открытых соединений, поскольку неизвестно, будет ли конкретный клиент вообще дальше взаимодействовать с источником данных. В объектной модели ADO.NET можно выделить несколько уровней (рисунок 3.1):

Уровень данных. Это базовый уровень, на котором располагаются сами данные (например, таблицы базы данных). На данном уровне обеспечивается физическое хранение информации на носителях и манипуляция с данными на уровне начальных таблиц (выборка, сортировка, добавление, удаление, обновление и т.п.).

Уровень бизнеса-логики. Это набор объектов, которые определяют, с какой базой данных необходимо установить связь и какие действия необходимо будет выполнить с содержащейся в ней информацией. Для установления связи с базами данных используется объект `DataConnection`. Для хранения команд, выполняющих действия над данными, используется объект `DataAdapter`. И, в конце концов, если выполнялся процесс выборки информации из базы данных, для хране-

ния результатов выборки используется объект DataSet. Объект DataSet по сути является набором данных, выбранных из таблиц основного хранилища, которые могут быть переданы любой программе-клиенту.

Уровень приложения. Это набор объектов, разрешающих хранить и отображать данные на компьютере конечного пользователя. Для хранения информации используется уже знакомый нам объект DataSet, а для отображения данных существует довольно большой набор элементов управления (DataGrid, TextBox, ComboBox, Label и т.п.). В Visual Studio .NET можно вести разработку двух типов приложений. В первую очередь это традиционные Windows-приложения (на основе Windows-форм), которые реализованы в виде exe-файлов, запускаемых на компьютере пользователя. Ну и конечно, Web-приложения (на основе Web-форм), работающие в оболочке браузера. Как видно из рисунка 3.1, для хранения данных на уровне приложений обеих типов используется объект DataSet.

В ADO.NET для работы с данными могут использоваться команды, которые реализованы в виде SQL-запросов или процедур.

Когда нужно получить набор строк из базы данных, необходимо выполнить следующую последовательность действий:

– открыть соединение (Connection) с базой данных;

– запустить на выполнение метод или команду, указав в качестве параметра текст SQL-запроса или имя хранимой процедуры;

– закрыть соединение с базой данных.

Соединение с базой данных остается активным только на период выполнения запроса или хранимой процедуры.

Когда команда запускается на выполнение, она возвращает или данные, или код ошибки. Если в команде содержался SQL-запрос на выборку SELECT, то команда может возвратить набор данных. Вы можете выбрать из базы данных только определенные строки и колонки, используя объект DataReader, который работает достаточно быстро, поскольку использует курсоры read-only (только для чтения) и forward-only (только для перемещения вперед).

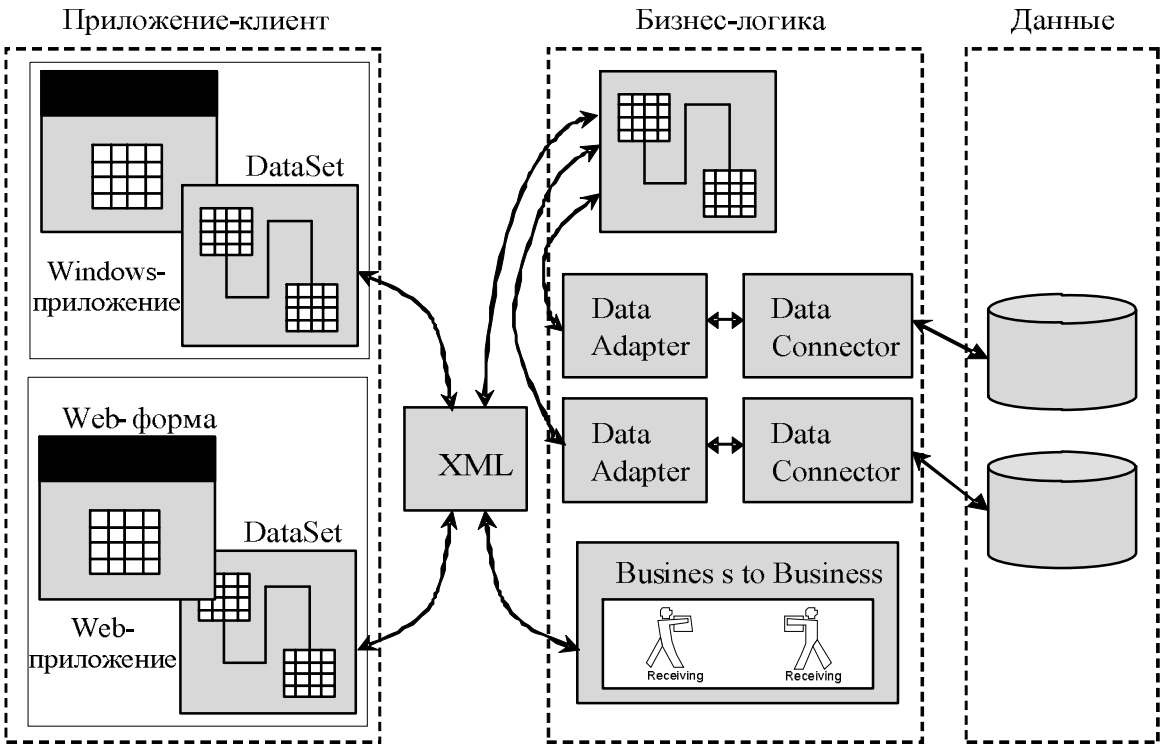

Рисунок 3.1 — Объектная модель ADO.NET

Как правило, в приложениях необходимо выбрать информацию из базы данных и выполнить с ней некоторые действия: показать пользователю на экране монитора, сделать необходимые расчеты или передать данные в другой компонент. Очень часто нужно обработать не одну запись, а их набор: список клиентов, перечень заказов, перечень позиций заказа и тому подобное. Как правило, в приложениях нужна одновременная работа с более чем одной таблицей: клиенты и все их заказы; автор и все его книги, заказ и его позиции, т.е. с набором связанных данных. Причем для удобства пользователя данные нужно группировать и сортировать по различным признакам. При этом нерационально каждый раза обращаться к исходной базе данных и заново перечитывать данные. Более практично работать с временным набором информации, хранящейся в оперативной памяти компьютера.

Эту роль выполняет набор данных — `DataSet`, являющийся своеобразным кешем записей, выбираемых из базового источника. В отличие от обычного `Recordset`, `Dataset` может состоять из одной или более таблиц, он имеет дело с копиями таблиц базы данных источника. Кроме того, в данном объекте могут содержаться связи между

таблицами и некоторые ограничения на выбираемые данные. Структура объекта `DataSet` приведена на рисунке 3.2.

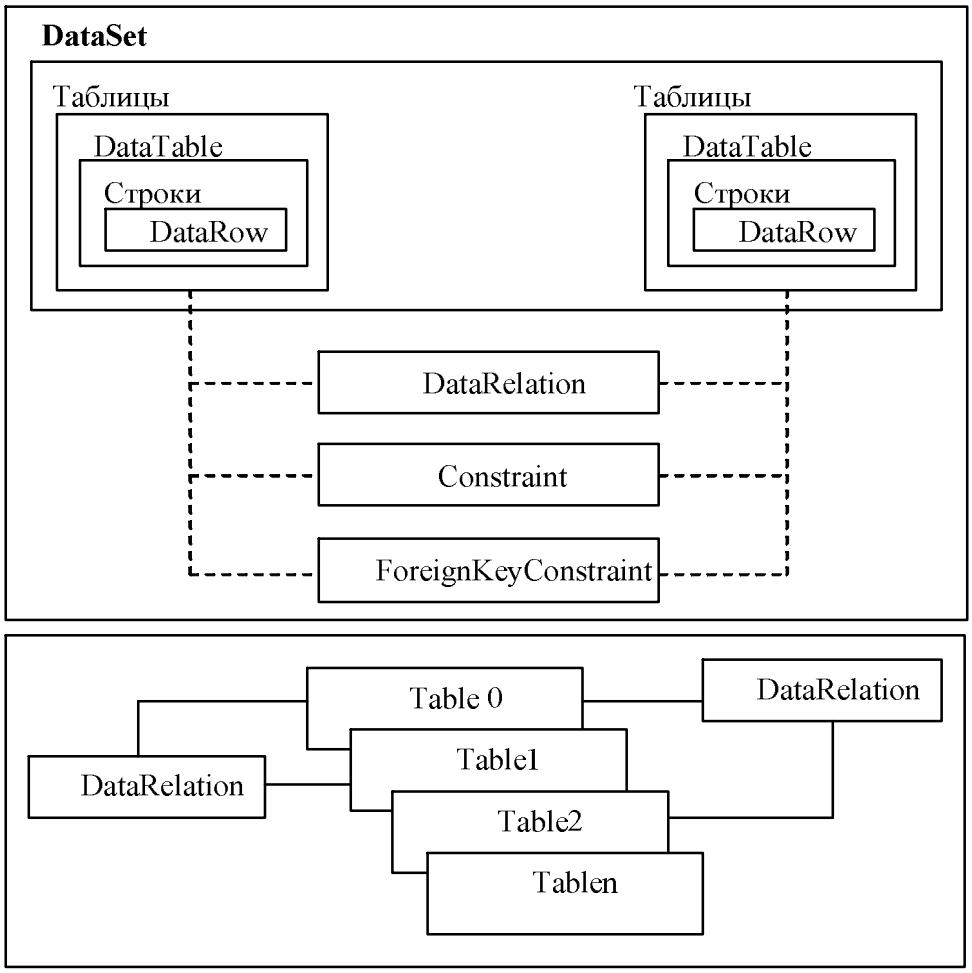

Рисунок 3.2 — Структура объекта `DataSet`

Данные в `DataSet` — это фрагмент основной базы данных, с которым Вы можете работать, как и с реальной базой. Поскольку каждый пользователь манипулирует с полученной порцией информации, оставаясь отсоединенным от основной базы данных, последняя может в это время выполнять другие задачи.

Конечно, практически в любой задаче обработки данных нужно корректировать информацию в БД (хотя и не так часто, как выбирать данные из нее). Вы можете выполнить операции коррекции непосредственно в `DataSet`, а потом все внесенные изменения будут переданы в основную базу данных.

Важно отметить то, что DataSet — пассивный контейнер для данных, обеспечивающий только их хранение. А что необходимо поместить в этот контейнер, определяется другим объектом — адаптером данных DataAdapter. В адаптере данных содержатся одна или несколько команд, определяющих, какую информацию нужно поместить в таблицы объекта DataSet, по каким правилам нужно синхронизировать информацию в конкретной таблице DataSet и соответствующей таблице основной базы данных и тому подобное. Адаптер данных обычно содержит четыре команды: SELECT, INSERT, UPDATE и DELETE для выборки, добавления, обновления и удаления записей.

Таким образом, набор данных DataSet — фрагмент базы данных, расположенный на компьютере пользователя. Причем в этой копии могут быть не отражены те изменения, которые могли внести в основную базу данных другие пользователи. Если нужно увидеть самые последние изменения, сделанные другими пользователями, то необходимо обновить DataSet, вызвав метод Fill адаптера данных.

3.2 Отображение информации из базы данных

3.2.1 Соединение с источником данных

ADO.NET-соединение представлено отдельным классом управляемого поставщика данных, отвечающим за соединение с источником данных. Поскольку ADO.NET ориентирован на работу с данными в отсоединенном режиме, класс соединения должен поддерживать повторяющиеся открытия соединения, и его закрытие.

Для соединения с сервером баз данных MySQL необходимо установить библиотеку mysql-connector-net. Для ее использования необходимо в свойствах проекта (меню «**Проект → Свойства»**) на вкладке **«Ссылка»** нажать кнопку «**Добавить»** и выбрать из списка библиотеку MySQL.Data. Также следует в коде проекта добавить следующую строку:

```
using MySql.Data.MySqlClient;
```

В коде, приведеном ниже, показан рекомендуемый жизненный цикл объекта Connection:

```
// Создание объекта соединения.
MySqlConnection conn = new MySqlConnection(
  “Server=localhost; Database=DB; ”  +
  “ user=root, pwd='root';”);

// Использование соединения
MySqlCommand cmd = new MySqlCommand(
  “SELECT COUNT (id_customer) FROM customers”, conn);

// Открытие соединения
conn.Open();

// Выполнение команды - получение данных
int result = Convert.ToInt32(cmd.ExecuteScalar());

// Закрытие соединения
conn.Close();

// Изменение команды
cmd.CommandText =
   “SELECT MAX(id_customer) FROM customers;”;

// Повторное открытие соединения
conn.Open();

// Выполнение команды - получение данных
double maxDiscount =
  Convert.To Double(cmd.ExecuteScalar());

// Закрытие соединения
conn.Close();
```

При создании объекта `Connection` определяется строка соединения, в которой описывается расположение источника данных (в нашем случае — локальный сервер баз данных). Потом соединение открывается и выполняется команда — запрос к базе данных ADO.NET. Как только получена нужная информация, соединение закрывается.

После обработки этой информации можно снова открыть соединение и продолжать работать с тем же источником данных. Обратите

внимание, что объект `Command` возможно использовать сколько угодно раз, поскольку необходимости в его создании для каждой операции нет.

В строке соединения с БД необходимо, как минимум, указать расположение базы данных и необходимую информацию авторизации (рис. 3.3). Кроме этого, каждый поставщик данных определяет дополнительные параметры соединения. Если в строке соединения не указаны значения всех возможных параметров, они считаются установленными по умолчанию.

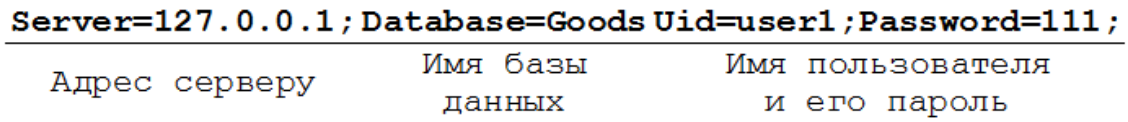

Рисунок 3.3 — Типовая строка подключения

Значение некоторых параметров строки соединения доступны через значение свойств, предназначенных только для чтения. В зависимости от того, определите Вы соответствующие параметры в строке соединения или нет, Вы сможете узнать их значение. Например, если в строке соединения значения параметра `ConnectionTimeout` не определено, Вы можете прочитать его в свойстве `MySqlConnection.ConnectionTimeout`.

Строка соединения, управляемого поставщиком MySQL Server, содержит множество параметров, но чаще всего используются только некоторые из них. Ниже перечислены наиболее часто используемые параметры:

– `Server` (Сервер). Имя или адрес компьютера с установленным сервером MySQL Server (например, `192.168.0.1`, `myserver.mydomain.com` или `localhost`);

– `Database` (База данных). Имя базы данных, находящейся на сервере (например `Pubs` или `Master`);

– `Integrated Security` (Встроенная система безопасности). Логический флаг, который указывает, будет SQL Server использовать NT-аутентификацию или выполнять проверку пары «имя пользователя/пароль». Установив значение параметра `Integrated Security` в `true`, Вы указываете серверу баз данных на необходимость использо-

вания текущего дескриптора NT Security для аутентификации доступа к базе данных. По умолчанию этот параметр равняется `false`;

– `Uid` или `User Name` (Идентификатор пользователя). Идентификатор, используемый для аутентификации пользователя на сервере баз данных (считается, что значение параметра `Integrated Security` равняется `false`);

– `Pwd` или `Password` (Пароль). Пароль, используемый для аутентификации пользователя на сервере баз данных (предполагается, что значение параметра `Integrated Security` равняется `false`);

– `Connection Timeout` (Продолжительность попытки установки соединения). Время в секундах, на протяжении которого будет осуществляться попытка соединения. Значение по умолчанию равно 15 с.

Во время жизненного цикла объекта `Connection` его состояние может меняться много раз, при изменении состояния возникают события. Обработка этих событий предусматривает регистрацию для получения сообщения о соответствующих событиях класса соединения (табл. 3.1).

Таблица 3.1 — События класса соединения

Событие	Описание
Disposed	Возникает при вызове метода Dispose. По обыкновению используется для высвобождения ресурсов, которые не могут быть освобождены автоматически
InfoMessage	Возникает, когда поставщик возвращает информационное или предостерегающее сообщения. Вызывается на усмотрение поставщика БД
StateChange	Возникает при открытии или закрытии соединения. Данное событие разрешает получить информацию о новом и старом состоянии соединения

3.2.2 Выполнение команд

В ADO. NET существует несколько способов выполнения команд, отличающихся лишь информацией, возвращаемой из базы данных. Ниже перечислены методы выполнения команд, которые поддерживаются всеми поставщиками БД:

– `ExecuteNonQuery()`. Как правило, этот метод применяется для выполнения команд, которые не возвращают результирующий набор данных. Так как при вызове метода `ExecuteNonQuery()` возвращается число строк, добавленных, измененных или удаленных в результате выполнения команды, которое может использоваться как индикатор успешного завершения операции;

– `ExecuteScalar()`. Этот метод выполняет команду и возвращает первый столбец первой строки первого результирующего набора данных. Метод `ExecuteScalar()` может быть полезным для получения аналитической информации из базы данных, например:

```
SELECT COUNT (userid) FROM USERS;
```

– `ExecuteReader()`. Этот метод выполняет команду и возвращает объект `DataReader` (детальнее он рассматривается в следующем разделе), который представляет собой однонаправленный поток записей из базы данных. Объект `Command` (команда) разрешает выполнять основные операции с базой данных: выборку, обновление, изменение и удаление.

Существует два основных способа создания объекта `Command`**:**

1. Непосредственно:

```
MySqlConnection conn  = new MySqlConnection();
MySqlCommand cmd = new MySqlCommand();
cmd.Connection = conn;
cmd.CommandText = "SELECT * FROM customers;";
```

2. На основе объекта `Connection`**:**

```
MySqlConnection conn  = new MySqlConnection();
MySqlCommand cmd = conn.CreateCommand();
cmd.CommandText = "SELECT * FROM customers;";
```

В последнем случае вместе с созданием нового объекта команды происходит определение соответствующего ему объекта соединения.

Так или иначе, требования «привязывать» соединение к определенной команде не существует.

Команды — это мощный инструмент, позволяющий проводить сложные операции с базой данных. В ADO.NET существует три типа команд:

1. `Text`. Текстовая команда состоит из инструкций, которые указывают поставщику БД на необходимость выполнения определенных действий на уровне базы данных. В большинстве случаев такая команда написана на SQL соответствующей базы данных (T-SQL для SQL Server, PL/SQL для Oracle и так далее). Обычно текстовые команды передаются в базу данных без предыдущей обработки (за исключением случаев передачи параметров). Этот тип команд поддерживается всеми поставщиками данных: SQL Server, OLE DB, Oracle и ODBC, MySQL.

2. `StoredProcedure`. Хранимая процедура — это команда, которая находится в самой базе данных. Такой тип команд поддерживается всеми поставщиками.

3. `TableDirect`. Команда типа `TableDirect` предназначена для получения из базы данных полной таблицы. Она аналогична текстовой команде `SELECT * FROM Имя_Таблицы`. Команда типа `TableDirect` поддерживается только поставщиком OLE DB.

Все команды, рассмотренные выше типы, можно выполнять по отношению к базе данных. По умолчанию свойство `CommandType` объекта команды установлено в значение `Text`. Для определения типа команды используется перечисление `CommandType`.

В коде ниже приведен пример вызова простой хранимой процедуры с помощью поставщика MySQL Server.

```
// Подключение к базе данных
MySqlConnection conn = new MySqlConnection(
    "Server=localhost;Database=master;" +
    "Integrated Security=true;");
conn.Open();

// Создание команды для выполнения
// хранимой процедуры
MySqlCommand cmd = conn.CreateCommand();
```

```
cmd.CommandText = “sp_stored_procedures”;
cmd.CommandType = CommandType.StoredProcedure;
// Определение параметров запроса
MySqlParameter param =
     new MySqlParameter(“?RETURN”, MySqlDbType.Int16);
param.Direction = ParameterDirection.ReturnValue;
cmd.Parameters.Add(param);

param = new MySqlParameter(“?sp_name”,
            MySqlDbType.VarChar);
param.Direction = ParameterDirection.Input;
cmd.Parameters.Add(param);

param = new MySqlParameter(“?sp_owner”,
            MySqlDbType.VarChar);
param.Direction = ParameterDirection.Input;
param.Value = “dbo”;
cmd.Parameters.Add(param);

param = new MySqlParameter(“?sp_qualifier”,
            MySqlDbType.VarChar;)
param.Direction = ParameterDirection.Input;
cmd. Parameters. Add (param);

// Выполнение хранимой процедуры
MySqlDataReader rdr = cmd.ExecuteReader();

// Перебор всех записей и вывод на консоль
// имен процедур хранимых процедур
while (rdr.Read())
Console.WriteLine(“Proc:”,rdr(“PROCEDURE_NAME”));

// Освобождение ресурсов
conn.Dispose();
```

Мы определили четыре параметра, которые необходимо передать в хранимую процедуру `sp_stored_procedure`. Как видим в коде, некоторые входные параметры не инициализированы — по умолчанию им будет присвоено значение `null` (в данном случае – `DBNull`). Другими словами, для того, чтобы присвоить параметру значения `null`, его

не нужно инициализировать. Для каждого параметра были заданы также несколько разных свойств. Некоторые из свойств параметров перечислены ниже (следует отметить, что не каждый параметр требует определения всех свойств):

– `ParameterName`. Имя параметра. Как правило, каждый разработчик баз данных имеет свои правила относительно именования параметров. Так, в поставщике MySQL перед именами параметров указывают символ «?». С другой стороны, большинство поставщиков определяют параметры по их позиции. В этом случае нужно создавать параметры в том порядке, в котором они передаются в хранимую процедуру или запрос с параметрами;

– `DbType`. Тип данных, которые хранятся в параметре. В .NET — перечислении `DbType` содержатся типы, которые можно определить для свойства `DbType`. Кроме того каждый поставщик имеет свойство, точнее реальный тип данных СУБД. Например, у поставщика SQL Server есть перечисление `SqlDbType`. При выборе специфического для поставщика элемента перечисления автоматически устанавливается соответствующее значение `DbType`. Именно так проводится отображение типов данных СУБД на типы поставщика. Например, установка свойства `DbType` параметра `SqlParameter` в значение `DbType.Boolean` приведет к автоматической установке свойства `SqlDbType` в значение `SqlDbType.Bit`;

– `Size`. Это свойство зависит от типа данных параметра и обычно используется для указания его максимальной длины. Например, для сроковых типов (`NVarChar`, `VarChar`, `Char`) свойство `Size` представляет максимальный размер строки. Значение по умолчанию определяется на основе типа `DbType` параметра — в большинстве случаев его можно использовать, не боясь возникновение каких-либо проблем. Определение максимального размера параметра способно уменьшить вероятность передачи процедуре сознательно неверных значений, длина которых превышает максимально допустимую (в этом случае СУБД сгенерирует ошибку);

– `Direction`. Данное свойство определяет способ передачи параметра. Его возможные значения: `Input`, `Output`, `InputOutput` и

ReturnValue (перечисление ParameterDirection). По умолчанию используется значение Input;

– isNullable. Это свойство определяет, может ли параметр принимать значение null. По умолчанию используется значение False;

– Value. Значение параметра. Для параметров типа Input или InputOutput это свойство должна быть установлено до выполнения команды, а для параметров типа InputOutput, Output и ReturnValue его значение устанавливается в результате выполнения команды. Чтобы передать пустой входной параметр, надо или не устанавливать значения свойства Value, или установить его значение равным DBNull. По умолчанию используется значение DBNull;

– Precision. Определяет число знаков (слева от десятичной точки), используемых для представления значения параметра. По умолчанию используется значение 0;

– Scale. Определяет число десятичных разрядов (число знаков справа от комы десятичной точки), используемых для представления значения параметра. По умолчанию используется значение 0;

– SourceColumn. Данное свойство определяет способ использования параметра с объектом DataAdapter;

– SourceVersion. Определяет, как и предыдущее свойство, способ использования параметра с объектом DataAdapter.

Объект Command также позволяет задавать запросы с параметрами. Рассмотрим один из наиболее простых вариантов создания запросов с параметрами. Предположим, что у нас есть запрос, который возвращает число записей, отвечающих определенному критерию:

```
SELECT COUNT(*) FROM customers WHERE city = 'Алчевск'
```

Этот запрос возвращает результирующий набор данных, который состоит из одной записи — количества строк, соответствующих заданному критерию. С помощью параметра можно динамически задавать этот критерий, например:

```
SELECT COUNT(*)FROM customers WHERE city = ?
```

Приведенный выше запрос позволяет узнать количество записей, которые отвечают критерию, определенному параметром запроса.

Параметры в запросах с параметрами используются таким же образом, как и в хранимых процедурах, с той лишь разницей, что вам не нужно создавать параметр для возвращаемого значения — в запросах с параметрами его просто нет.

Следует отметить одну важную особенность — формат запросов с параметрами отличается для разных поставщиков. В таблице 3.2 приведенный соответствующий синтаксис для различных комбинаций поставщиков и баз данных.

Таблица 3.2 — Синтаксис параметров для различных поставщиков и БД

Поставщик	База данных	Синтаксис параметра
SQL Server	SQL Server	? или @Имя_параметра
OLE DB	Oracle	? или @Имя_параметра
OLE DB	MS Access	? или @Имя_параметра
ODBC	SQL Server	?Имя_параметра
MySQL	MySQL	?Имя_параметра

Использование запросов с параметрами демонстрируется ниже.

```
// Создание команды, содержащей запрос с параметрами
MySqlCommand cmd = conn.CreateCommand()
cmd.CommandText =
  "SELECT * FROM customers Where id_customer=?CustID";

// Определение параметра
cmd.Parameters.Add(
  "?CustID", MySqlDbType, Int16).Direction =
    ParameterDirection.Input;
cmd.Parameters("?CustID").Value = Guid.NewGuid();
```

3.2.3 Чтение данных и объект DataReader

После того, как Вы научились создавать запросы, пришло время научиться их выполнять, используя для этого средства ADO.NET. В отличие от большинства других интерфейсов доступа к данным, в которых для чтения и изменения данных используется один и тот же объект, в ADO.NET операции чтения и изменения данных разделены.

В частности, большая часть кода просто будет считывать информацию из базы данных, и будет представлять ее пользователю, используя для этого основной объект ADO.NET, предназначенный исключительно для чтения данных — объект DataReader. Он обеспечивает доступ к данным, извлекаемым в результате выполнения SQL-запроса, в режиме только для чтения. Объект DataReader представлен соответствующим классом — OleDbDataReader, SqlDataReader, OracelDataReader, MySqlDataReader и OdbcDataReader — для каждого из поставщиков. Пример использования класса MySqlDataReader представлен ниже.

```
// Создание объекта команды.
MySqlCommand cmd = conn.CreateCommand();
cmd.CommandText = "SELECT * FROM customers";

// Создание объекта DataReader
// в результате выполнения запроса
MySqlDataReader rdr = cmd.ExecuteReader();

// Итерация по всем строкам.
while (rdr.Read()) {
  Console.WriteLine(rdr(0));
}
```

В данном примере приведен наиболее простой способ использования объекта DataReader**.** Для перемещения по результирующему набору данных предназначен метод Read(). Обратите внимание, что метод Read() обязательно должен быть вызван еще до чтения первой записи. Благодаря этому объект DataReader более стойкий к ошибкам, поскольку перемещение курсора и проверка того, достигнут ли конец результирующего набора данных, производится в одном операторе. По достижению конца результирующего набора данных метод Read() возвратит False.

Объект DataReader не может быть создан непосредственно из клиентского кода — он создается объектом команды во время ее выполнения с помощью метода ExecuteReader().

Как было показано раньше, объект DataReader разрешает осуществлять доступ к отдельным столбцам, представляя результирующий набор данных в виде массива. Пример доступа к столбцам результирующего набора данных с помощью объекта DataReader представлен ниже.

```
// Создание объекта команды.
MySqlCommand cmd = conn.CreateCommand();
cmd.CommandText = "SELECT * FROM customers";

// Создание объекта DataReader
// в результате выполнения запроса.
MySqlDataReader rdr = cmd.ExecuteReader();

// Итерация по всем строкам.
while (rdr.Read()) {
  Console.WriteLine(rdr(0));
  Console.WriteLine(rdr("id_customer "));
  Console.WriteLine(rdr.GetString(0));
}
```

Три оператора, размещенные в теле цикла, имеют одинаковое назначение. Первый оператор считывает столбец результата с номером 0, как имеющий тип Object. Второй оператор считывает столбец результата по имени CUSTOMERID, третий — считывает столбец результата с номером 0, как имеющий тип String.

3.2.4 Объекты DataSet и DataAdapter

Объект DataSet можно представить, как:

– набор информации, которая извлечена из базы данных, доступ к которой осуществляется в отключенном режиме;

– база данных, которая расположена в оперативной памяти;

– сложная реляционная структура данных со встроенной поддержкой XML-сериализации.

Объект DataSet состоит из нескольких связанных одна с другой структур данных. Концептуально он является полным набором реляционной информации.

Внутри объекта DataSet могут храниться пять типов объектов:

– DataTable — набор данных, организованный в виде столбцов и строк. Этот объект является аналогом объекта ADO Recordset и объекта OLE DB RowSet;

– DataColumn — коллекция правил, описывающая данные, которые можно сохранять в объектах DataRow;

– DataRow — коллекция данных, представленная одной строкой таблицы DataTable. Объект DataRow является фактическим хранилищем данных;

– Constraint – правило, задающее допустимость хранения определенных данных в объекте DataTable**.** Объект Constraint используется для определения бизнесов-правил для объекта DataSet;

– DataRelation – описание навигационных связей между различными объектами DataTable. В большинстве случаев объект DataRelation сопровождается объектом Constraint, что разрешает строго задать отношение.

Как показано на рисунке 3.4, внутри объекта DataSet может храниться коллекция объектов DataTable**.** В свою очередь внутри объекта DataTable хранятся коллекции объектов DataRow**,** DataColumn, Constraint и две коллекции объектов DataRelation.

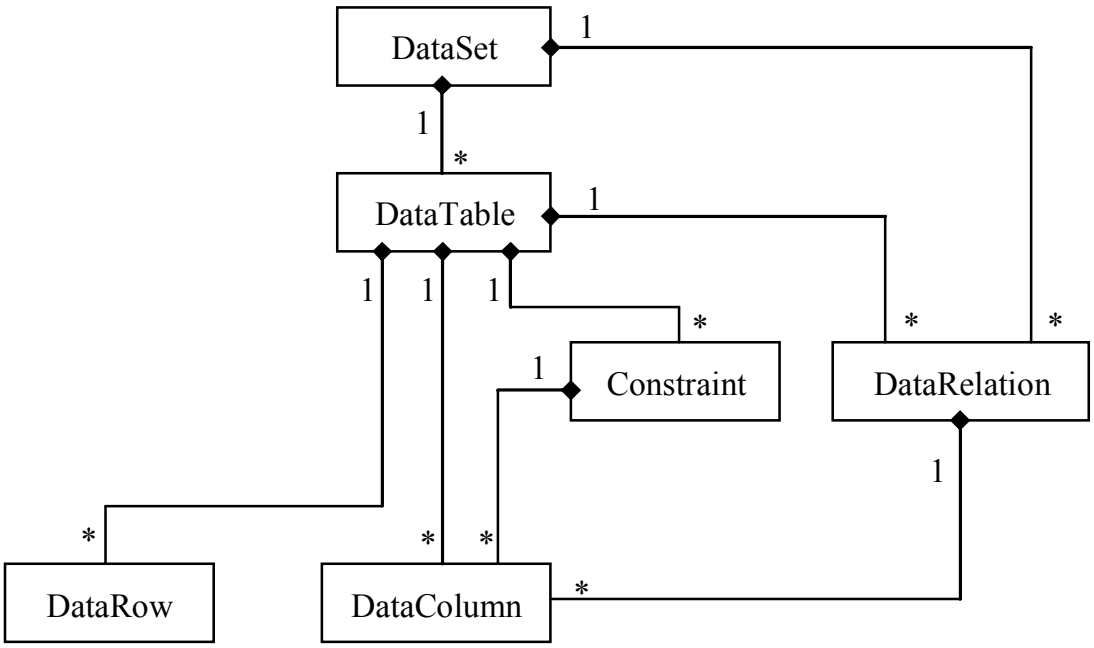

Рисунок 3.4 — Структура объекта DataSet

Коллекция объектов DataRelation формирует связь между объектами DataTable на уровне «родитель-потомок». Коллекция объектов DataRelation внутри объекта DataSet является агрегированным представлением всех объектов DataRelation во всех объектах DataTable.

Поскольку объект DataSet не связан с базой данных, его можно воспринимать как реляционное хранилище данных. Для того чтобы заполнить объект DataSet, совсем не обязательно использовать информацию, которая хранится в базе данных. На практике используют три метода заполнения объекта DataSet: с помощью объекта DataAdapter (при этом информация, как правило, извлекается из базы данных), на основе XML-документа и вручную.

Объект DataAdapter является объединяющим звеном между объектом DataSet и хранилищем данных. С помощью объекта DataAdapter можно заполнять объект DataSet информацией из хранилища данных, а также обновлять хранилище данных на основе объекта DataSet**.** Объект DataAdapter состоит из четырех объектов Command, каждый из которых выполняет строго определенную задачу:

– SelectCommand — используется для извлечения данных;

– InsertCommand — используется для добавления новых записей, созданных в объекте DataSet;

– UpdateCommand — используется для обновления существующих записей на основе изменений в объекте DataSet;

– DeleteCommand – используется для удаления существующих записей на основе удалений в объекте DataSet.

Объект DataAdapter используется каждый раз, когда объект DataSet нуждается в непосредственном взаимодействии с источником данных. Объект DataAdapter содержит объекты Command для каждой из основных операций, выполняемых над базой данных: SELECT, INSERT, UPDATE и DELETE. Основное назначение объекта DataAdapter состоит в формировании связи между объектом DataSet и базой данных.

В простейшем случае объект DataSet используется для хранения одной таблицы, как показано ниже.

```
// Создание объекта Command для получения
// из базы данных таблицы Customer.
MySqlCommand cmd = conn.CreateCommand();
cmd.CommandText = "SELECT * FROM customers";

// Создание объекта DataAdapter
// для заполнения объекта DataSet.
MySqlDataAdapter dataAdapter =
               new MySqlDataAdapter(cmd);

// Создание пустого объекта DataSet
DataSet dataSet = new DataSet();

// Заполнение объекта DataSet
// с помощью объекта DataAdapter
dataAdapter.Fill(dataSet);
```

В данном примере объект DataSet заполняется результирующим набором данных из таблицы customers.

Преимущества объекта DataSet проявляются при его использовании как реляционного хранилища данных, расположенного в памяти. Рассмотрим пример использования пакетных запросов для получения данных из нескольких таблиц.

```
// Создание строки-запроса query
// для того, чтобы выбрать из базы данных
// несколько таблиц, используем пакетный
// запрос, который возвращает
// несколько результирующих наборов данных.
String query = "";
query +- "SELECT * FROM customers;";
query += "SELECT * FROM services;";
query += "SELECT * FROM workers;";

// Создание объекта DataAdapter
// для выборки данных
```

```
MySqlDataAdapter dataAdapter =
      New MySqlDataAdapter(query, conn);

// Создание и заполнение объекта DataSet
DataSet dataSet = new DataSet();
dataAdapter.Fill(dataSet);
```

3.2.5 Объект DataView

Класс `DataView` используется для привязки данных к элементам управления Windows-форм и Web-форм в ASP.NET. Каждый объект `DataTable` содержит объект `DataView`. Объект `DataView`, используемый по умолчанию: `DataTable.DefaultView`. Можно создавать столько объектов `DataView`, сколько необходимо. Это может оказаться полезным при отображении одних и тех же данных, отфильтрованных или отсортированных различным образом, как показано в следующем коде:

```
// Создание переменной для доступа к таблице
DataTable dataTable = dataSet.Tables[0];

// Подсчет количества строк в представлении, которое
// используется по умолчанию
Console.WriteLine("DefaultView Count: {0}",
        dataTable.DefaultView.Count);

// Создание нового объекта DataView
DataView sortedView = new DataView(dataTable);
sortedView.Sort = "LastName";
```

В данном примере используются два разных представления — принятое по умолчанию, и представление `sortedView`. Они никак не влияют на данные, но содержат на них разную «точку зрения».

Класс `DataView` поддерживает сортировку данных с помощью свойства `DataView.Sort`. Свойство `Sort` является строкой, описывающей критерий сортировки. Для того, чтобы выполнить сортировку с помощью объекта `DataView`, просто установите значения свойства `Sort`, использовав для этого имя одного или нескольких столбцов с мо-

дификатором ASC или DESC. По умолчанию используется возрастающий порядок сортировки, поэтому его можно не указывать, как показано в следующем коде:

```
// Создание переменной для доступа к таблице
DataTable dataTable = dataSet.Tables[0];

// Сортировка по LastName (по возрастанию).
dataTable.DefaultView.Sort = "city";

// Сортировка по LastName (по убыванию).
dataTable.DefaultView.Sort = "city DESC";

// Сортировка по LastName (по возрастанию)
// а потом по FirstName (по убыванию)
dataTable.DefaultView.Sort =
                    "name_customer, city DESC";
```

Класс DataView поддерживает фильтрацию данных с помощью свойства DataView.RowFilter. Свойство RowFilter является строкой, описывающей критерий фильтрации. Пример фильтрации данных с помощью объекта DataView показан в следующем коде:

```
// Создание переменной для доступа к таблице
DataTable dataTable = dataSet.Tables[0];

// Определение фильтра
dataTable.DefaultView.RowFilter =
                "city = 'Алчевск'";

// Подсчет строк, которые удовлетворяют фильтру
Console.WriteLine("RowFilter count: {0}",
            dataTable.DefaultView.Count);
```

Рассмотрим несколько примеров фильтров.

Текстовый фильтр:

```
dataTable.DefaultView.RowFilter =
                          "city = 'Алчевск'";
```

В качестве критерия фильтрации можно использовать дату, которую необходимо заключить в символы диеза (#):

```
dataView.RowFilter = "date_begin = #04/24/1976#";
```

Операторы сравнения:

```
dataView.RowFilter = "payment > 3200.00";
```

Поддерживаются операторы сравнения <, >, <= и >=.

Сложные выражения:

```
dataView.RowFilter =
  "(payment <= 4500.00) AND (Cost > 100.00)";
```

Для составных выражений поддерживаются булева конкатенация (`AND`, `OR` и `NOT`) и задание приоритетов с помощью скобок.

Групповые символы:

```
dataView.RowFilter =
        "city LIKE 'A*'";
```

В рамках синтаксиса `LIKE` для текстового сравнения поддерживаются групповые символы «*» и «%». Приведенный выше фильтр выведет всех заказчиков, место расположения города которых начинаются с «А», независимо от того, сколько еще символов находится после первого.

3.2.6 Связанные элементы управления

Большинство элементов управления вкладки Windows Forms имеют встроенную возможность отображать в форме информацию из базы данных. Элемент управления называется связанным с источником данных, если его свойство `DataBindings` содержит имя одного из полей набора данных. Элементы управления, которые могут отображать информацию из базы данных: `TextBox`, `ComboBox`, `ListBox`, `CheckBox`, `RadioButton`, `DataGridView`, `PictureBox`.

Связать с данными можно разные свойства элементов. В элементе `TextBox` чаще всего связывают с данными свойство `Text`. Например,

следующий код свяжет свойство Text текстового поля tb1 с полем IDCustomer таблицы Customers из набора данных ds:

```
tb1.DataBindings.Add("Text",
ds.Tables["customers"],  "id_customer");
```

Для связывания с базой данных элемента DataGridView достаточно в его свойстве DataSource указать нужный объект DataTable:

```
DGV.DataSource = ds.Tables["customers"];
```

Для перемещения между записями DataSet используется свойство Position. Например, в обработчике события нажатия кнопки, отображающей первую запись таблицы Customer из набора данных DataSet, необходимо записать:

```
this.BindingContext(ds, "customers").Position = 0;
```

Общее количество записей содержится в свойстве Count. Например, чтобы перейти на последнюю запись, необходимо записать:

```
this.BindingContext(ds, "customers").Position =
   this.BindingContext(ds, "customers").Count - 1;
```

3.3 Модификация данных

3.3.1 Модификация объекта DataSet с использованием связанных элементов управления

Если Вы используете для отображения данных элемент DataGridView, то модификация данных в нем автоматически изменяет данные в DataSet. Не нужно осуществлять никаких дополнительных действий, необходимо только переслать эти изменения в базу данных.

Но это не является действительным для связанных текстовых полей. Если Вы просто отредактируете значение в текстовом поле, то это никак не отобразится в наборе данных. Для сбора данных со связанных

текстовых полей используется экземпляр библиотечного класса System.Windows.Forms.BindingContext. Объект BindingContext содержится в любом элементе управления и разрешает управлять процессом редактирования и обновление данных, а также их передачей в набор данных с помощью метода EndCurrentEdit():

```
this.BindingContext(ds, "customers").EndCurrentEdit();
```

Этот код необходимо поместить в обработчик события фиксации внесенных изменений, после чего можно отправлять изменения в базу данных.

Для отмены изменений используйте метод CancelCurrentEdit(), отменяющим все изменения, которые проводились в текущем текстовом поле.

Для добавления новой строки с помощью связанных текстовых полей необходимо вызвать метод:

```
this.BindingContext(ds, "customers").AddNew();
```

Этот метод добавит новую строку в набор данных и отобразит ее в текстовом поле для заполнения. После ввода данных вызовите метод EndCurrentEdit() для фиксации строки в DataSet.

Для удаления текущей строки используйте метод RemoveAt:

```
int index = this.BindingContext(dataSet,
            "customers").Position;
this.BindingContext(dataSet,
            "customers").RemoveAt(index);
```

Если необходимо отменить все изменения в объекте DataSet, которые были осуществлены после последней связи с базой данных, используйте метод RejectChanges() объекта DataSet:

```
ds.RejectChanges();
```

3.3.2 Программная модификация данных в DataSet

Большинство систем управления базами данных разрешают определить набор действий, которые выполняются при изменении, добавлении или удалении информации. Например, при добавлении в базу данных нового потребителя может сработать триггер, который внесет этого потребителя в таблицу поиска. Объект `DataSet` поддерживает регистрацию для получения сообщений о проведении определенных операций над объектами `DataTable`, что позволит выполнять соответствующий код обработки. Ниже перечислены типы событий, которые поддерживаются объектом `DataSet`:

– `RowChanging` — вызывается перед изменениями строк объекта `DataTable`;

– `RowChanged` — вызывается после изменения строк объекта `DataTable`;

– `ColumnChanged` — вызывается после изменений данных в столбцах;

– `RowDeleting` — вызывается перед удалением строки из объекта `DataTable`;

– `RowDeleted` — вызывается после удаления строки из объекта `DataTable`.

Событие добавления строки присутствует неявно, поскольку ADO.NET рассматривает любые изменения свойства `Rows` (включая добавление новой строки) как предпосылку для срабатывания событий `RowChanged` и `RowChanging`. События `RowChanging`, `RowChanged`, `RowDeleting` и `RowDeleted` предоставляют ссылки на измененную, удаленную или добавленную строку и объект `DataRowChangeEventArgs`, который содержит значение из перечисления типа `DataRowAction`. Последнее можно использовать для выяснения того что же происходит со строкой. Ниже приведены возможные значения этого перечисления:

– `Add` — строка была добавлена в таблицу `DataTable`;

– `Change` — строка была изменено;

– `Commit` — измененная строка была внесено в базу данных;

– Delete — строка была удалено;

– Nothing — строка не была изменена;

– Rollback — произошла отмена последних внесенных в строку изменений.

Перечисление DataRowAction дает возможность точно определить, что произошло со строкой. Поскольку объект DataRowChangeEventArgs используется для всех событий строк, можно ожидать, что события RowChanging и RowChanged сработают и при удалении строки. На самом деле, это не так — при удалении строк срабатывают только события RowDeleting и RowDeleted.

Основным назначением объекта DataSet является хранение и изменение данных. Независимо от того, был он создан в результате чтения информации из базы данных или программным путем, объект DataSet обязательно содержит данные.

Объекты DataRow является основным хранилищем данных внутри объекта DataSet**.** Объект DataRow содержит массив значений, которые являются строкой объекта DataTable**.** Объекты DataRow доступные из объекта DataTable через свойство Rows, являются коллекцией объектов DataRow (для этого используется класс DataRowCollection). Объект DataRow отличается от традиционных коллекций полей, поскольку он, кроме текущего значения каждого столбца в строке, включает в себя некоторую дополнительную информацию:

– текущее значение каждого столбца в строке;

– исходное значение каждого столбца в строке, если оно было изменено;

– состояние столбца (добавленного, удаленного, измененного);

– средство перехода к родительской или дочерней строке (при наличии отношений).

Ниже приведен пример изменения данных в наборе данных ds:

```
ds.Tables[“customers”].Rows(4).Item(1)= “Грин Хаус”;
```

Эта команда изменит имя потребителя в пятой строке второго поля таблицы Customers на «Грин хаус».

Добавлять новые строки к объекту DataTable довольно просто. Вы можете или создать массив значений и добавить его к коллекции строк как новую строку или создать новую строку с помощью объекта DataTable, как показано в следующем коде:

```
// Создание переменной для доступа к таблице
DataTable dataTable = dataSet.Tables[0];

// Добавление новой строки путем
// создание объекта DataRow
DataRow newRow = dataTable.NewRow();
newRow("id_customer")= Guid.NewGuid();
newRow("name_customer")= "Планета";
newRow("city")= "Алчевск";
newRow("address")= "ул. Абаканская 90";

dataTable.Rows.Add(newRow); // <- Важно!

// Добавление новой строки путем создания
// массива значений столбцов
Object newValues = new
           Object(dataTable.Columns.Count);

newValues(0)= Guid.NewGuid();
newValues(1) = "Планета";
newValues(2)= "Брянка";
newValues(3)= "ул. Абаканская 90";

dataTable.Rows.Add(newValues); // <- Важно!
```

Метод DataRowCollection.Add() поддерживает оба способа добавления новой строки, однако лучше использовать метод DataTable.NewRow(), поскольку он разрешает заполнить новую строку значениями, гарантируя их соответствие требованиям столбцов. Другими словами, значение в новом объекте DataRow удовлетворяют схеме объекта DataTable, что делает невозможным добавление некорректных значений. И вдобавок явное обращение к элементам строки сделает код более читабельным. Действительно, первый способ добав-

ления нового объекта DataRow более прозрачен, поскольку можно увидеть, что нулевым столбцом является идентификатор клиента, первый столбец — его именем, а третий столбец — фамилией.

Следует заметить, что метод DataTable.NewRow() сам по себе не добавляет строку в объект DataTable. Для этого необходимо вызвать метод DataTable.Rows.Add(), передав ему в качестве параметра объект строки.

Новые строки не обязательно добавляются в конец коллекции строк объекта DataTable. Если Вам необходимо вставить новую строку в середину коллекции, воспользуйтесь методом DataTable.Rows.InsertAt(), как показано в следующем коде:

```
// Добавление новой строки
// путем создания объекта DataRow
DataRow insertedRow = dataTable.NewRow();
insertedRow("id_customer")= Guid.NewGuid();
insertedRow("name_customer")= "Планета";
insertedRow("city")= "Алчевск";
insertedRow("address")= "ул. Абаканская 90";
dataTable.Rows.InsertAt(insertedRow, 1);
```

При использовании отсоединенных данных к удалению строки из коллекции предъявляется особое требование: строка должен продолжать существовать до тех пор, пока хранилище данных не будет обновлено с помощью объекта DataSet. Удаление строки может быть осуществлено одним из трех способов:

– с помощью метода DataTable.Rows.Removed на основе заданного экземпляра класса DataRow;

– с помощью метода DataTable.Rows.RemoveAt() на основе заданного порядкового номера строки;

– с помощью метода DataRow.Delete() (строка удаляет себя сама).

Рассмотрим некоторые особенности удаления строк. Удаление строки может вызвать изменение порядковых номеров строк. Если добавить и удалить строку прежде, чем изменения будут приняты объектом DataTable, то система фактически удалит элемент из коллекции.

При удалении элемента, существующего за пределами объекта DataTable, строка сохраняется, но обозначается как удаленная — информация, которая хранится в строке, еще понадобится для проведения обновления хранилища данных. Таким образом, использование порядкового номера строки очень ненадежно, поскольку он может изменяться каждый раз при удалении или добавлении элементов. Использование рассмотренных способов удаления строк показано в следующем коде:

```
// Создание переменной для доступа к таблице
DataTable dataTable = dataSet.Tables[0];
// Добавление строки путем создания
// нового объекта DataRow
DataRow newRow = dataTable.NewRow();
newRow("id_customer")= Guid.NewGuid();
newRow("name_customer")= "Планета";
newRow("city")= "Алчевск";
newRow("address")= "ул. Абаканская 90";
dataTable.Rows.Add(newRow); // <- Важно!
// Удаление строки на основе заданного экземпляра
dataTable.Rows.Remove(newRow);
// Создание переменной для доступа к первой строке
DataRow row = dataSet.Tables[0].Rows[0];
// Удаление строки
row.Delete();
// Удаление строки на основе порядкового номера
dataTable.Rows.RemoveAt(2);
```

Вывод данных, хранящихся в объекте DataSet, может быть осуществлен путем перебора объектов DataRow заданного объекта DataTable, как показано в следующем коде:

```
// Создание переменной для доступа к таблице
DataTable dataTable = dataSet.Tables[0];

// Вывод имени каждого клиента на консоль
Console.WriteLine("Customer List");
Console.WriteLine("-------------");
for each (DataRow dr in DataTable.Rows) {
  Console.WriteLine(
```

```
    "{0}, {1}", dr("name_customer"), dr("city"));
}
```

3.3.3 Сохранение изменений в базе данных

Сохранение изменений в базе данных происходит при помощи метода `Update()` объекта `DataAdapter`. Напомним, что объект `DataAdapter` содержит команды (объекты типа `Command`) для выполнения операций вставки, обновления и удаления информации из базы данных. Метод `Update()` поочередно просматривает каждую строку и в зависимости от значения ее свойства `RowState` выполняет соответствующую команду объекта `DataAdapter` (таблица 3.3).

Таблица 3.3 — Правила выбора команды `DataAdaper`

Значение RowState	**Команда DataAdapter**
Modified (изменена)	UpdateCommand
Deleted (удалена)	DeleteCommand
Added (добавлена)	InsertCommand

Команды можно сформировать автоматически с помощью специального объекта `CommandBuilder` (построитель команд):

```
MySqlCommandBuilder custBldr =
            new MySqlCommandBuilder(dataAdapter);
```

Кром того, можно формировать отдельные команды:

```
dataAdapter.getUpdateCommand();
dataAdapter.getDeleteCommand();
dataAdapter.getInsertCommand();
```

Здесь `dataAdapter` — имя объекта `DataAdapter`, используемого для доступа к таблице БД.

Часто бывает необходимо вручную формировать запросы для модификации базы данных. Прежде чем создать собственные команды, необходимо понять, как команды объекта `DataAdapter` используют параметры при определении обновляемых данных. Каждая из этих команд имеет набор параметров, которые представляют столбцы базы

данных. Параметры объекта Объект DataAdapter обеспечивают правильное соединение между объектом DataRow и базой данных. Ниже перечислены важные свойства классов параметров (OleDbParameter, SqlParameter, MySqlParameter, OracleParameter и OdbcParameter):

– SourceColumn — имя столбца в таблице DataTable, которому соответствует этот параметр;

– SourceVersion — версия (DataRowVersion) строки DataRow, которой соответствует этот параметр;

– Direction — это свойство имеет особое значение, поскольку после выполнения команды значения входного (Output) или входного/выходного (InputOutput) параметра будет помещено в столбец SourceColumn строки DataRow.

Для создания собственных команд обновления необходимо провести их настройку. Первые два свойства параметра (SourceColumn и SourceVersion) привязывают его к определенному столбцу. Свойство SourceVersion предназначено для обозначения версии строки, с которой необходимо связать параметр. В общем случае рекомендуется связывать с версией DataRowVersion.Current параметры, предназначенные для фактического обновления информации в базе данных, а с версией DataRowVersion.Original – параметры, предназначенные для использования в предикате WHERE с целью идентификации строки.

По умолчанию свойство Direction имеет значение Input, тем не менее, Вам, вероятно, нужно будет изменить его, создавая параметр, используемый для возврата значения из базы данных (например, идентификационного номера для строки для вставки в таблицу). По желанию Вы можете построить оператор INSERT так, чтобы столбец первичного ключа был исходным параметром, разрешая тем самым SQL-оператору или хранимой процедуре установить его (параметра) значение после вставки строки в базу данных.

Рассмотрим пример создания команд INSERT, UPDATE и DELETE:

```
// Создание команды INSERT
```

```
String insQry;
insQry = "INSERT INTO customers(";
insQry += "id_customer, name_customer, city, address";
insQry += "VALUES (?id_customer, ?name_customer, ";
insQry += "?city, ?address)";
MySqlCommand insCmd = conn.CreateCommand();
insCmd.CommandText = insQry;

// Создание переменной для упрощения
// доступа к коллекции параметров.
SqlParameterCollection insParams =
                            insCmd.Parameters;

// Определение параметров
insParams.Add("?id_customer", MySqlDbType.Int16,
               0, "id_customer");
insParams.Add("?name_customer", MySqlDbType.VarChar,
               50, "name_customer");
insParams.Add("?city", MySqlDbType.VarChar,
               50, "city");
insParams.Add("?address", MySqlDbType.VarChar,
               50, "address");
insParams("?address").IsNullable = true;

// Установка значения свойства
// InsertCommand объекта DataAdapter
dataAdapter.InsertCommand = insCmd;

// Создание команды DELETE
String delQry;
delQry = "DELETE FROM customers ";
delQry += "WHERE id_customer=?id_customer";
MySqlCommand  delCmd = conn.CreateCommand();
delCmd.CommandText = delQry;

// Определение переменной для упрощения
// доступа к коллекции параметров
MySqlParameterCollection delParams =
                 delCmd.Parameters;

// Определение параметров
```

```
delParams.Add("?id_customer", MySqlDbType.Int16,
          2, "id_customer");
delParams("?id_customer").SourceVersion =
          DataRowVersion.Original;

// Установка значения свойства DeleteCommand
// объекта DataAdapter
dataAdapter.DeleteCommand = delCmd;

// Создание команды UPDATE
String updQry;
updQry = "UPDATE customers SET ";
updQry += "name_customer = ?name_customer,";
updQry += "city = ?city,";
updQry += "address= ?address";

updQry += "WHERE id_customer=?id_customer";
SqlCommand updCrnd = conn.CreateCommand();
updund.CommandText = updQry;

// Определение переменной для упрощения
// доступа к коллекции параметров
MySqlParameterCollection updParams =
              updCmd.Parameters;

// Определение параметров
updParams.Add("?id_customer",
MySqlDbType.Int16, 2, "id_customer");
updParams("?id_customer").SourceVersion =
          DataRowVersion.Original;

updParams.Add("?name_customer",
MySqlDbType.VarChar, 50, "name_customer");

updParams.Add("?city", MySqlDbType.VarChar,
               50, "city");

updParams.Add("?address", MySqlDbType.VarChar,
               50, "address");
updParams("?Address").IsNullable = true;
```

```
// Установка значения свойства UpdateCommand
// объекта DataAdapter
dataAdapter.UpdateCommand = updCm;
```

3.4 Работа с объектами dataset, состоящими из нескольких таблиц

3.4.1 Создание объекта DataSet

В большинстве случаев приходится иметь дело с объектами DataSet, состоящих из нескольких таблиц. Основное отличие бъект DataSet, содержащий одну таблицу, от объекта DataSet, содержащего несколько таблиц, заключается в том, что каждой таблице объекта DataSet должен отвечать собственный объект DataAdapter.

```
// Создание объектов DataAdapter
MySqlDataAdapter custDA = new MySqlDataAdapter(
       "SELECT * FROM customers", conn);

MySqlDataAdapter ordDA = new MySqlDataAdapter(
      "SELECT * FROM orders", conn);

// Создание объекта DataSet
DataSet dataSet = new DataSet();

// Использование объекта DataAdapter
// для заполнения объекта DataSet
custDA.Fill(dataSet, "customers");
ordDA.Fill(dataSet, "orders");

// Определение переменной для доступа к таблице
DataTable custTable = dataSet.Tables["customers"];
DataTable ordTable = dataSet.Tables["orders"];

// Определение отношения
DataColumn custCustIDColumn =
               custTable.Columns["id_customer"];

DataColumn ordCustIDColumn =
    ordTable.Columns["id_customer"];
```

```
dataSet.Relations.Add("rel", custCustIDColumn,
    ordCustlDColumn, true);
```

3.4.2 Отображение данных из DataSet на форме

Рассмотрим пример отображения данных из объекта DataSet, содержащего таблицы `orders` (Заказы) и `customers` (Потребители), связанных по полю `CustomerID`.

Если подключить к компоненту `DataGridView` таблицу `Orders`, то в поле `CustomerID` будет отображаться номер потребителя, который осуществил данный заказ. Конечно, это будет создавать сложность при идентификации потребителя во время просмотра таблицы, а также в процессе добавления новых записей, когда пользователь должен сначала узнать код нужного потребителя, а потом ввести его в поле. Кроме того, в случае невыполнения условия целостности данных (например, введен код потребителя, которого нет в таблице `customers`) будет сгенерирована ошибка. Более удобной является возможность выбирать из списка коды доступных потребителей, а еще удобнее — выбирать, например, фамилии. Рассмотрим, как это реализовать.

Для этого необходимо добавить в таблицу столбец типа `DataGridViewComboBoxColumn` и задать значения таких свойств (табл. 3.4).

Таблица 3.4 — Установление значений свойств столбца `DataGridViewComboBoxColumn`

Свойство	Значение
DataSource	Таблица, содержащая поле, из которого будет формироваться список подстановки
DataPropertyName	Поле, значение которого фактически содержится в поле
ValueMember	Первичный ключ таблицы, из которой осуществляется подстановка
DisplayMember	Поле, из которого берутся значение для отображения в списке

Например, следующий код настраивает элемент управления `DataGridView` с именем `DGV` на отображение таблицы заказов покупателей `Orders`, добавляет в таблицу новый столбец типа `DataGridViewComboBoxColumn`, и настраивает его таким образом, чтобы

в нем вместо кода сотрудника IDCustomer отображался список фамилий покупателей (LastName):

```
DGV.DataSource = ds.Tables[“orders”];
DataGridViewComboBoxColumn cbc =
       new DataGridViewComboBoxColumn();
cbc.DataSource = ds.Tables[“workers”];
cbc.DataPropertyName = “id_worker”;
cbc.ValueMember = “id_worker”;
cbc.DisplayMember = “FIO”;
```

Поле, вместо которого мы создаем новое поле со списком, можно сделать невидимым:

```
DGV.Columns[“id_worker”].Visible = false;
```

В результате получим удобную форму просмотра и редактирование таблицы “Orders” (рис. 3.5).

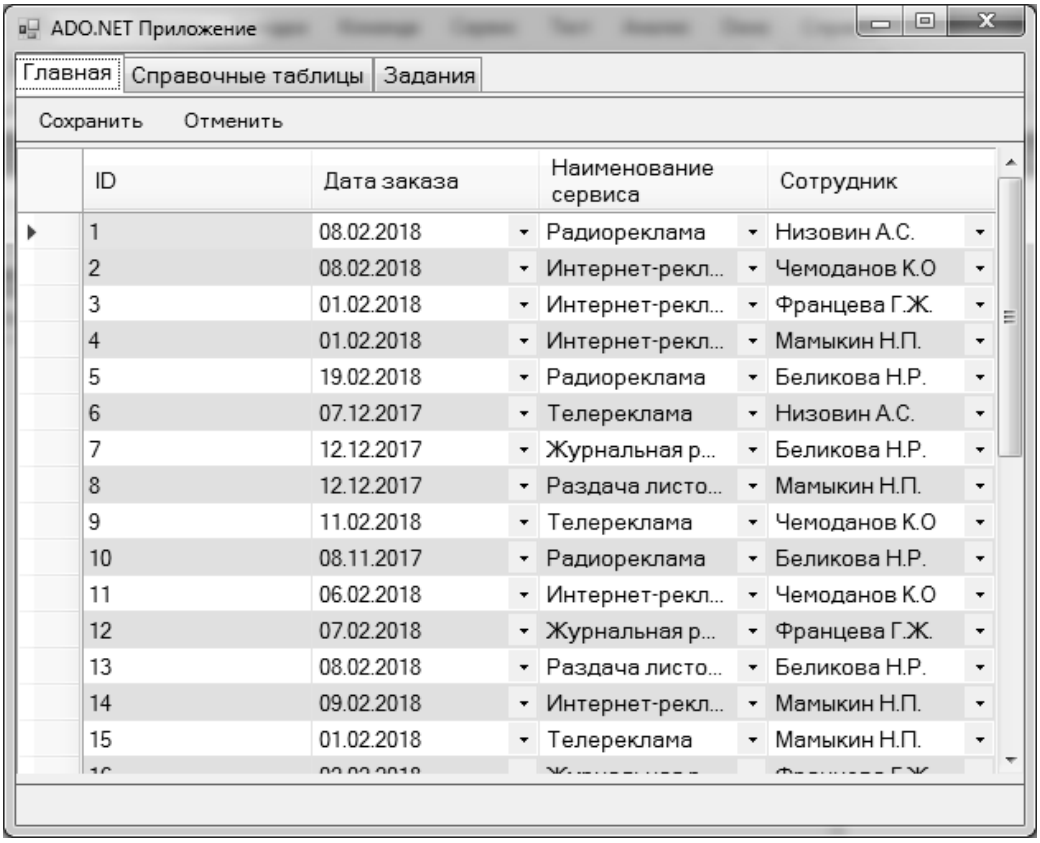

Рисунок 3.5 — Форма просмотра таблицы “Orders”

3.5 Создание отчетов

Visual C# .NET предлагает несколько компонентов, с помощью которых можно вывести пользователю отчет:

– MicrosoftReportViewer;

– CrystalReportViewer;

– WebBrowser.

Первые два компонента имеют более обширный набор функций являются более удобными при условии создания их с помощью визуальных средств проектирования. Однако Visual Studio .NET не предоставляет возможности создавать источник данных для БД MySQL средствами визуального проектирования, а программная настройка указанных компонентов является довольно сложной.

В этих условиях проще всего использовать компонент WebBrowser, который позволяет динамически формировать текст web-страницы, выполняющей функции отчета. Достаточно лишь считать необходимые данные из базы данных (рациональнее всего с помощью объекта DataReader), сформировать текст страницы и передать его в свойство DocumentText объекта WebBrowser.

В следующем фрагменте кода приведен пример формирования отчета:

```
String html="";
MySqlCommand cmd =
  new MySqlCommand(task, conn.getConnection());
MySqlDataReader reader = cmd.ExecuteReader();

int count = reader.FieldCount;
html += "<style type=\"text/css\">\n" +
  "TABLE {\n" +
  "    border-collapse: collapse;\n" +
  "    background: #232323;\n" +
  "    font-family: sans-serif;\n" +
  "    color: white;\n" +
  "   }\n" +
  "   TD, TH {\n" +
  "    padding: 5px;\n" +
  "    border: 1px solid #BD6F00;\n" +
```

```
  "    }" +
  "  </style>\n" +
  "  <body bgcolor = \"#232323\">\n" +
  "  <table cellspacing=\"0\" bgcolor = \"white\">"; ;
  html += "<tr style = \"background: #303030;\">";
for (int i = 0; i < count; i++)
  html += "<td>" + reader.GetName(i) + "</td>";

html += "</tr>";
while (reader.Read())
{
  html += "<tr>";
  for (int i = 0; i < count; i++)
    html += "<td>" + reader.GetString(i) + "</td>";
  html += "</tr>";
}
html += "</table>";
webBrowser1.DocumentText = html;
reader.Close();
```

На рисунке 3.6 приведен сформированный отчет.

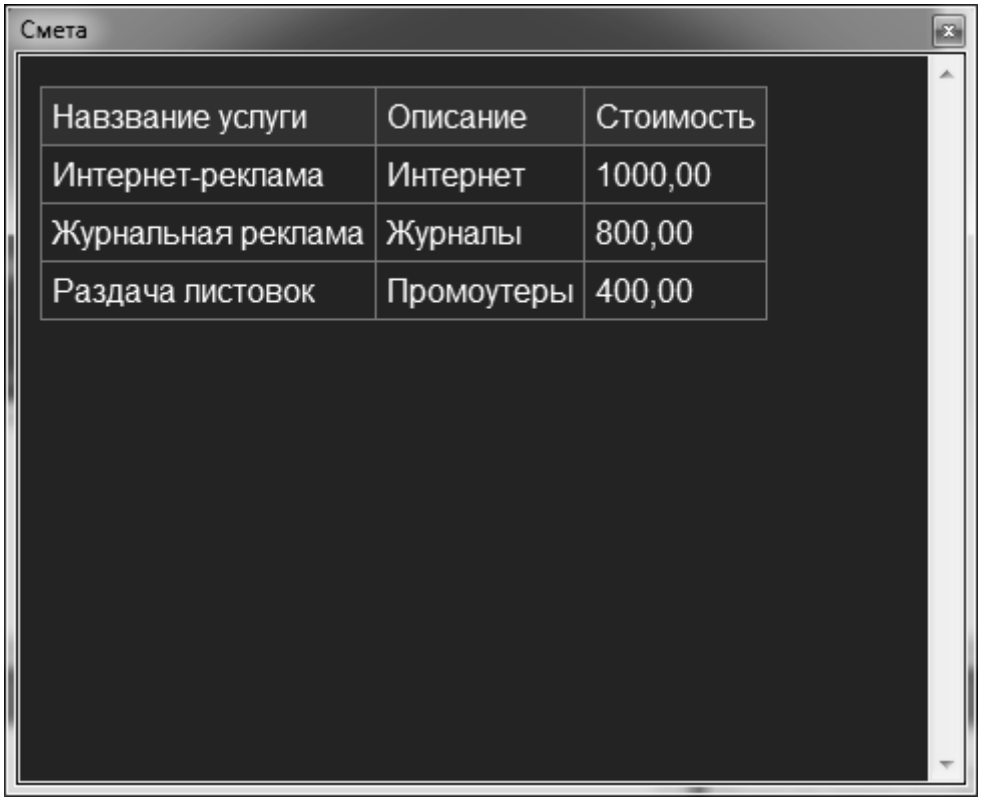

Рисунок 3.6 — Окно отчета

3.6 Задание к лабораторной работе

Разработайте приложение для управления БД, которая была создана Вами при выполнении первой и второй лабораторных работ.

1. Доступ к справочным таблицам должен осуществляться через связанные элементы управления `TextBox`. Для доступа к основной таблице используйте элемент управления `DataGridView`. Для навигации и управление записями таблиц БД установите кнопки `Button` (переход на первую строку, предыдущая строка, следующая строка, последняя строка, добавление записи, удаление записи, фиксация изменений, отмена изменений).

2. Обеспечьте возможность вывода результатов запросов, которые были созданы Вами в лабораторной работе №2, в компонент `DataGridView`.

3. Используя результаты третьего запроса, создайте отчет на основе компонента `WebBrowser`. Отчет должен выводиться на отдельной форме.

4. Для управления работой программы предусмотрите панель инструментов и главное меню. Для вывода подсказок используйте строку состояния и флажки.

5. Запрограммируйте приложение. Обеспечьте условия для минимизации ошибок пользователя. Обеспечьте кеширование изменений. Создайте систему предупреждений для пользователя в случае возникновения ошибки.

6. Обеспечьте печать данных из таблиц и отчета.

3.7 Требования к отчету

Отчет оформляется на стандартных листах белой бумаги формата А4 (210х297 мм). Текст пишется с одной стороны листа чернила синего или черного цвета или печатается на принтере.

Бланк отчета готовится во время домашней подготовки к работе и должен содержать:

– данные о студенте: фамилия и инициалы, шифр группы;
– тему и цель работы;
– дату выполнения работы;

– набор данных, необходимых для выполнения работы: задача к работе, код программы, короткие теоретические сведения о функциях и библиотеках Visual C# .NET, которые использовались при подготовке отчета.

После выполнения работы к отчету подшиваются: распечатка программного кода, распечатка результатов.

Отчет должен быть выполнен аккуратно. Неаккуратно выполненные отчеты, а также ксерокопии чужих отчетов или их фрагментов к защите не допускаются.

3.8 Контрольные вопросы

1. Перечислите уровни объектной модели ADO.NET.
2. Опишите жизненный цикл соединения с базой данных MySQL.
3. Параметры строки соединения с базой данных. Назначение параметра `Integrated Security`.
4. Какие Вы знаете события объекта `Connection`?
5. Назначение объекта `Command`.
6. Чем отличаются методы `ExecuteNonQuery()`, `ExecuteScalar()`, `ExecuteReader()`?
7. Какие Вы знаете способы создания объекта `Command`?
8. Поддерживает ли объект `Command` команды с параметрами?
9. Что определяет свойство `Direction` параметра команды?
10. Как установить значение параметра?
11. Назначение свойства `SourceVersion`.
12. Синтаксис параметров для СУБД MySQL?
13. Какой объект ADO.NET предназначенный исключительно для чтения данных?
14. Назначение объекта `DataAdapter`.
15. Как привязать текстовое поле к полю таблицы базы данных?
16. Назначение и структура объекта `DataSet`.
17. Методы заполнения объекта `DataSet`.
18. Может ли объект `DataSet` содержать несколько таблиц БД?
19. Как осуществить фильтрацию данных в объекте `DataSet`?

20. Какие элементы управления могут отображать информацию из базы данных?

21. Как метод `Update` объекта `DataAdapter` определяет, какую команду (`INSERT`, `UPDATE`, `DELETE`) применить к измененной строке?

22. Как установить связь между таблицами в объекте `DataSet`?

23. Как настроить элемент управления `DataGridView` на отображение данных таблицы БД?

24. Какие компоненты предлагает Visual C# .NET для создания отчетов?

25. С помощью какого свойства компоненту `WebBrowser` задается текст страницы отчета?

4 ЛАБОРАТОРНАЯ РАБОТА № 4

Тема: программирование сетевых баз данных в Visual C# .NET.

Цель работы: научиться создавать сетевые приложения с использованием технологии ASP.NET для работы с базами данных.

4.1 Привязка данных в ASP.NET

При разработке Web- и ASP.NET-приложений большое значение имеет отсоединенная природа протокола HTTP. Привязка данных к элементам управления Web-форм осуществляется в режиме только для чтения. Это не похоже на привязку данных к элементам управления Windows-форм, характеризующихся непосредственной связью с источником данных. Иначе говоря, привязка данных в ASP.NET более удобна для размещения данных в элементах управления Web, чем для связывания элементов управления с данными.

4.1.1 Простая привязка данных

В ASP.NET простая привязка данных к элементам управления осуществляется с помощью синтаксиса на странице ASP.NET, с помощью следующего синтаксиса:

```
<asp:Textbox id="theTextBox"
text='<%# customers.Rows(0)("name_customer") %>'
runat="server" />
```

ASP.Net-страница обратится к таблице `customers`, выберит из нее значение столбца `name_customer` первой строки и поместит его в свойство `Text` элемента управления `TextBox`.

Для того чтобы завершить привязку данных, необходимо применить метод `DataBind()` элемента управления `TextBox`:

```
theTextBox.DataBind()
```

В этом и заключается принципиальная разница между привязкой данных к элементам управления Windows-форм и Web-форм. Элементы управления Windows-форм связываются с реальными данными (изме-

нения в элементе управления служат причиной изменений в данных), в то время как элементы управления Web-форм связываются с копией данных.

Вызов метода `DataBind()` элемента управления, размещенного на ASP.NET-странице, приведет к фактической привязке данных к этому элементу управления. Для того, чтобы связать с данными все элементы управления ASP.NET-страницы, следует вызвать метод `Databind()` для всей страницы.

Привязка данных осуществляется ядром ASP.NET путем создания кода привязки и его размещение в фоновом коде страницы.

В программном коде страницы привязка данных к текстовому полю ввода будет выглядеть так:

```
target.Text = _Convert.ToString(
   productTable.Rows(0)("name_customer"))
```

Здесь `target` – имя элемента управления (свойство `Name`).

Для того чтобы привязка данных прошла успешно, необходимо сделать таблицу `customers` членом фонового кода страницы.

4.1.2 Сложная привязка данных

Как и в Windows-формах, целью сложной привязки данных является связывание коллекции или списка элементов с элементом интерфейса пользователя — чаще всего с элементами `ListBox` или `DropDownList`. Эти элементы управления (и другие, подобные) поддерживают следующие свойства:

- `DataSource` — объект данных, с которым будет связан элемент управления. Обычно это объект `DataTable` или `DataView`;
- `DataTextField` — поле объекта, указанного в свойстве `DataSource`, которое будет использовано для отображения в элементе управления;
- `DataTextFormatString` — необязательная строка форматирования, разрешающая элементу управления определять формат столбца, указанного в свойстве `DataTextField`;

– DataValueField — поле объекта, указанного в свойстве DataSource, в котором сохраняется значение элемента управления.

Рассмотрим пример сложной привязки данных к элементам управления Web-форм.

```
// Привязка данных к элементу Listbox
theListBox.DataSource = Session("customerTable");
theListBox.DataTextField = "name_customer";
theListBox.DataValueField = "id_customer";
theListBox.DataTextFormatString = "Name: {0}";

// Привязка данных к элементу DropDownList
theDropDown.DataSource = Session("servicesTable");
theDropDown.DataTextField = "services";
theDropDown.DataValueField = "id_service";

// Привязка данных ко всем элементам управления
DataBind();
```

В ASP.NET существует несколько элементов управления, разрешающих отображать более одного столбца таблицы — DataGrid, DataList и Repeater. Привязка данных к этим элементам осуществляется так же, как и к элементу управления DataGrid Windows-форм — достаточно установить значение свойства DataSource и необязательного свойства DataMember:

```
// Привязка таблицы productTable к DataGrid
theGrid.DataSource = dataSet;
theGrid.DataMember = "services";
DataBind();
```

Как только привязка осуществлена, можно определить специфические элементы управления для привязки к ним определенных столбцов или разрешить им автоматически генерировать столбцы или строки.

4.1.3 Привязка с использованием DataReader

Привязка данных в ASP.NET осуществляется в режиме только для чтения, элементы управления Web-форм можно связать с объектом `DataReader`, как показано ниже:

```
// Создание соединения
MySqlConnection conn;
conn.Connectionstring =
  "Server='localhost';Database='DB';UserID='root';" +
  "Password='root'";
conn.Open();

// Создание объекта команды
MySqlCommand cmd;
cmd = conn.CreateCommand();
cmd.CommandText =
"SELECT name_service, id_service FROM services";

// Создание объекта DataReader для привязки
// к элементу управления ListBox
MySqlDataReader rdr;
rdr = cmd.ExecuteReader();

// Привязка данных к элементу ListBox
theListBox.DataSource = rdr;
theListBox.DataTextField = "name_service";
theListBox.DataValueField = "id_service";
theListBox.DataBind();

// Закрытие соединения
conn.Close();
```

Привязка данных с использованием объекта `DataReader` оправдана при работе с данными, которые часто изменяются (т.е. когда кеширование является неэффективным).

4.1.4 Кеширование данных

Возможность привязки данных в ASP.NET имеет большое значение, однако способ хранения данных за пределами Web-страницы игра-

ет очень важную роль, фактически, определяя скорость ее работы. При этом обычно рассматривается вопрос о необходимости кеширования и, если решение принимается в пользу кеширования — о месте его проведения.

Большинство ASP-узлов даже не пытаются кешировать данные, допуская, что база данных имеет достаточный запас производительности для мгновенного предоставления информации для каждой страницы. Иногда это оказывается довольно эффективным решением. Если разрабатывается приложение для локальной сети или небольшого Web-узла, использование объектов `DataReader` без кеширования является приемлемым решением.

Если из-за размера или масштаба Web-узла кеширование становится необходимостью, решением может стать кеширование на стороне сервера. Для хранения некоторой информации в памяти с целью обеспечения ее доступности для ASP-страниц были созданные целые системы. ASP.NET предоставляет большое число вариантов выбора кеширования. В отличие от ASP-страниц, в ASP.NET для хранения данных, которые относятся к определенному пользователю, применяется система управления состоянием сеансов (Session State Management System). ASP.NET поддерживает состояние сеанса локального уровня, уровня Web-кластера и уровня SQL Server, так что механизм поддержки состояния сеанса найдется для каждой ситуации.

Для кеширования данных, не относящихся к конкретному пользователю (например, списка товаров), можно применить статический класс (класс, все члены которого статические) или сохранить информацию на уровне приложения. Кеширование на уровне приложения имеет смысл применять в случае доступности для всех пользователей одной и той же информации. Недостатоком такого подхода заключается в том, что при изменении данных они меняются для всех сеансов одновременно.

4.2 Манипулирование данными в web-приложениях

Манипулирование данными состоит в выполнении следующих действий и операций: добавление новых записей, удаление существующих записей и обновление существующих записей. Рассмотрим последовательно эти операции.

4.2.1 Добавление записей

Добавить новые строки к таблице можно, используя объект DataTable. Рассмотрим это на примере:

```
// Создание переменной типа DataTable
DataTable datatable = dataset.Tables(0);
// Добавление красной строки путем создания
// объекта DataRow
DataRow newrow = datatable.Newrow();
newrow("id_customer") = Guid.NewGuid();
newrow("name_customer")= "Планета";
newrow("city")= "Алчевск";
newrow("address")= "ул. Абаканская 90";
datatable.Rows.Add(newrow);
```

Другой способ добавления — использование объекта MySqlCommand:

```
// Создание соединения.
MySqlConnection conn conn.Connectionstring =
  "Server='localhost';Database='DB';UserID='root';" +
  "Password='root'";
conn.Open();

// Создание объекта команды
MySqlCommand cmd;
cmd = conn.CreateCommand();
cmd.CommandText = "INSERT INTO " +
    "customers(name_customer, city, address) " +
    "VALUES('Планета', 'Алчевск', 'ул. Ленина 9')";
cmd.ExecuteNonQuery();
```

4.2.2 Удаление записей

При использовании отсоединенных данных для операции удаления строки из коллекции предъявляется особое требование: строка должна продолжать существовать до тех пор, пока хранилище данных не будет обновлено с помощью объекта DataSet. Удаление строки может быть осуществлено одним из четырех способов:

– с помощью метода DataTable.Rows.Remove() на основе заданного экземпляра класса DataRow;

– с помощью метода DataTable.Rows.RemoveAt() на основе заданного порядкового номера строки;

– с помощью метода DataRow.Delete() (строка удаляет себя сама);

– с использованием объекта MySqlCommand.

Все эти способы можно использовать для удаления заданных строк. При этом важно понимать, что удаление строки может вызвать изменение порядковых номеров строк. Если добавить и удалить строку перед тем, как изменения будут сохранены объектом DataTable, то система фактически удалит элемент из коллекции. При удалении элемента, существующего за пределами объекта DataTable, строка сохраняется, но помечается как удаленная — информация, хранимая в строке, еще понадобится для выполнения обновления хранилища данных.

Использование порядкового номера строки очень ненадежно, так как он может изменяться каждый раз при удалении или добавлении элементов. Разные способы удаления строки показаны ниже в листинге.

```
// Удаление строки для заданного экземпляра
datatable.Rows.Remove(newrow);
// Строка для доступа к первому записи таблицы
 DataRow row = dataset.Tables(0).Rows(0);
// Удаление строки
row.Delete();
// Удаление на основе порядкового номера
datatable.Rows.RemoveAt(2);
// Удаление с использованием MySqlCommand
cmd.CommandText = "DELETE FROM customers " +
                  "WHERE id_customer = 10";
cmd.ExecuteNonQuery();
```

4.2.3 Обновление записей

Обновление записей в таблице базы данных можно сделать двумя способами.

Первый использует методы BeginEdit (начать редактирование) и EndEdit (завершить редактирование) объекта DataRow. Второй способ состоит в выполнении оператора UPDATE объектом MySqlCommand. Рассмотрим пример:

```
// Создание соединения.
MySqlConnection conn conn.Connectionstring =
   "Server='localhost';Database='DB';UserID='root';" +
   "Password='root'";
conn.Open();

//  -------------- Первый способ --------
// Определение переменной для доступа к строке
DataRow row = dataSet.Tables(0).Rows(0);

// Начинаем редактирование
row.BeginEdit();
// Установка значений
row("name_customer") = "Сантех Плюс";
row("city") = "Алчевск";
// Завершаем редактирование
row.EndEdit();

//  -------------- Второй способ --------
// Создание объекта команды
MySqlCommand cmd;
cmd = conn.CreateCommand();
cmd.CommandText = "UPDATE customers " +
  "SET name_customer = 'Сантех Плюс', " +
  "city = 'Алчевск' " +
  "WHERE id_customer = 16";
cmd.ExecuteNonQuery();
```

4.2.4 Транзакции в ADO.NET

Транзакция – это атомарное действие, выполненное относительно база данных.

Одна из наиболее распространенных проблем программирования взаимодействия с базами данных связанная с необходимостью обеспечения общего выполнения нескольких дискретных операций. Например,

если Вы разрабатываете систему управления медицинскими карточками, и Ваша база данных не смогла сохранить информацию о новом пациенте, сумев при этом сохранить сведения о его визите, система будет находиться в несогласованном состоянии. Транзакция базы данных разрешает создать оболочку для набора операций, которые или выполняются все вместе, или не выполняются вообще.

Транзакция представлена специальным объектом — экземпляром класса `MySqlTransaction`. Ниже приведен пример управления транзакцией:

```
MySqlConnection conn conn.Connectionstring =
   “Server=‘localhost’;Database=‘DB’;UserID=‘root’;” +
   “Password=‘root’”;
conn.Open();
MySqlCommand cmd = conn.CreateCommand();

//Создание транзакции
cmd.Transaction = conn.BeginTransaction();
try {
  // Попытка добавление в БД нового клиента
  cmd.CommandText = “INSERT INTO ” +
    “customers(name_customer, city, address) ” +
    “VALUES(‘Планета’, ‘Алчевск’, ‘ул. Ленина 9’)”;
  end.ExecuteNonQuery();
  // Попытка добавление в БД нового заказа
  cmd.CommandText = “INSERT INTO orders” +
    “ (date_begin, date_end, id_customer) ” +
    “ VALUES('2018-02-08', '2018-04-18', newID())”;
  cmd. ExecuteNonQuery();
  // Если оператор выполнено без ошибок,
  // подтверждаем транзакцию

  cmd.Transaction.Commit();
  // При возникновении ошибки - отменяем транзакцию
catch (Exception ex) {
  cmd.Transaction.Rollback()
  Console.WriteLine (“Ошибка в команде: ” +
  ex.Message + “Transaction Rolled back”); }
}
```

Транзакция создается путем вызова метода `BeginTransaction` класса соединения. В результате вызова метода создается объект транзакции. После этого ни одна из команд, ассоциированных с данным соединением, не будет подтверждена до вызова метода класса транзакции `Commit`. В случае возникновении ошибки для отмены всех команд, выполненных с момента создания транзакции, можно вызвать метод `Rollback`.

После начала транзакции ее нужно принять или отклонить**.** Если этого не сделать, то во время освобождения ресурсов транзакции она будет автоматически отклонена. Иначе говоря, транзакция автоматически отклоняется при закрытии соединения или освобождении занятых им ресурсов, а также при уничтожении объекта транзакции соборщиком мусора.

4.3 Пример web-приложения

4.3.1 База данных проекта и настройка проекта

Создадим базу данных Agency (Агенство) с такой структурой:

```
CREATE DATABASE IF NOT EXISTS Agency
CHARACTER SET utf8
COLLATE utf8_general_ci;
USE Agency;

-- таблица заказчики
CREATE TABLE customers (
  id_customers int(11) NOT NULL AUTO_INCREMENT
    COMMENT 'ключ заказчика',
  name_customer varchar(50) NOT NULL
    COMMENT 'наименование заказчика',
  address varchar(80) NOT NULL
    COMMENT 'адрес заказчика',
  passport varchar(20) NOT NULL
    COMMENT 'номер паспорта/свидетельства регистрации',
  PRIMARY KEY (id_customers)
)

ENGINE = INNODB
AUTO_INCREMENT = 16
AVG_ROW_LENGTH = 1092
CHARACTER SET utf8
```

```
COLLATE utf8_general_ci;

-- таблица оказанных услуг (смета)
CREATE TABLE estimate (
  id_estimate int(11) NOT NULL AUTO_INCREMENT
    COMMENT 'ключ оказанной услуги',
  id_order int(11) NOT NULL
    COMMENT 'внешний ключ на заказ',
  id_service int(11) NOT NULL
    COMMENT 'внешний ключ на услугу',
  id_worker int(11) NOT NULL,
  PRIMARY KEY (id_estimate),
  CONSTRAINT id_order_fk FOREIGN KEY (id_order)
    REFERENCES orders (id_order),
  CONSTRAINT id_service_fk FOREIGN KEY (id_service)
    REFERENCES services (id_service),
  CONSTRAINT id_worker_fk FOREIGN KEY (id_worker)
    REFERENCES workers (id_worker)
)
ENGINE = INNODB
CHARACTER SET utf8
COLLATE utf8_general_ci;
-- таблица заказов
CREATE TABLE orders (
  id_order int(11) NOT NULL AUTO_INCREMENT
    COMMENT 'ключ заказа',
  date_begin date NOT NULL
    COMMENT 'дата начала заказа',
  date_end datetime DEFAULT NULL
    COMMENT 'дата окончания заказа',
  id_customers int(11) NOT NULL
    COMMENT 'внешний ключ на заказчика',
  payment tinyint(1) NOT NULL
    COMMENT 'признак факта оплаты',
  PRIMARY KEY (id_order),
  CONSTRAINT id_customers_fk FOREIGN KEY (id_customers)
  REFERENCES customers (id_customers)
)
ENGINE = INNODB
CHARACTER SET utf8
COLLATE utf8_general_ci;
```

```
-- таблица предоставляемых услуг
CREATE TABLE services (
  id_service int(11) NOT NULL AUTO_INCREMENT,
  name_service varchar(50) NOT NULL,
  description varchar(255) DEFAULT NULL,
  cost decimal(19, 2) NOT NULL,
  PRIMARY KEY (id_service)
)
ENGINE = INNODB
CHARACTER SET utf8
COLLATE utf8_general_ci;

-- таблица работников
CREATE TABLE workers (
  id_worker int(11) NOT NULL AUTO_INCREMENT
    COMMENT 'ключ работника',
  FIO varchar(80) NOT NULL
    COMMENT 'ФИО работника',
  birthday date NOT NULL
    COMMENT 'дата рождения',
  sex tinyint(1) UNSIGNED NOT NULL DEFAULT 0
    COMMENT 'пол работника 0 - М, 1 - Ж',
  address varchar(80) DEFAULT NULL
    COMMENT 'адрес работника',
  phone bigint(20) DEFAULT NULL
    COMMENT 'номер телефона',
  PRIMARY KEY (id_worker)
)
ENGINE = INNODB
CHARACTER SET utf8
COLLATE utf8_general_ci;
```

Схема базы данных представлена на рисунке 4.1.

Для возможности работы с базой данных MySQL загрузите и установите последнюю версию MySQL Connector .NET с сайта www.mysql.com (раздел Downloads(MySQL Connectors)).

Создайте новый проект типа `ASP.NET Web Application`. В окне **Solution Explorer** нажмите правую клавишу мыши на названии проекта и выберите команду **Add Reference** (Добавить ссылку). В от-

крывшемся окне, выберите из списка библиотеку `MySQL.Data` и нажмите кнопку «ОК».

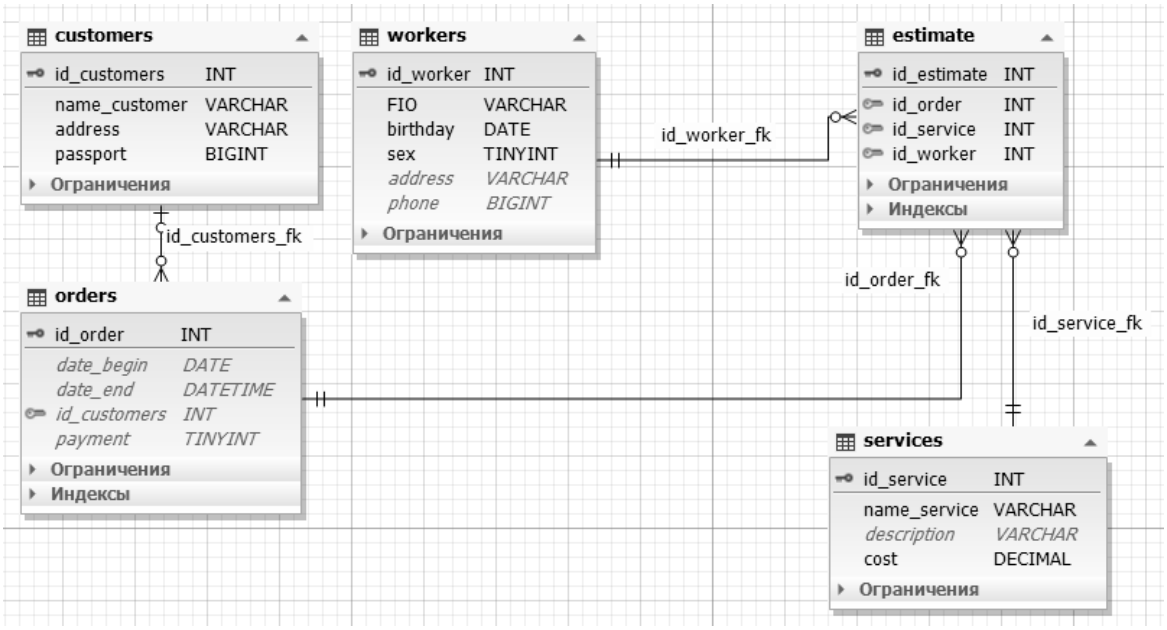

Рисунок 4.1 — Схема базы данных Agency

4.3.2 Создание форм приложения

Приложение содержит 8 форм (Web-страниц):

– главная форма `Default.aspx`;

– форма с информацией о приложении `About.aspx`;

– форма работы со справочником «Заказчики» (таблица БД customers) `Customers.aspx`;

– форма работы со справочником «Услуги» (таблица БД services) `Services.aspx`;

– форма работы со справочником «Сотрудники» (таблица БД workers) `Workers.aspx`;

– форма работы с таблицей «Заказы» (таблица БД orders) `Orders.aspx`;

– форма работы с таблицей «Смета» (таблица БД estimate) `Estimate.aspx`;

– форма для вывода отчетов `Tasks.aspx`.

После создания шаблонов форм окно **Solution Explorer (Обозреватель решений)** будет выглядить, как на рисунке 4.2.

Помимо этого в приложении присутствует ряд вспомогательных классов, упрощающих работу с БД, данные классы находятся в каталоге `Classes`. Некоторые из данных классов будут рассмотрены ниже.

На рисунках 4.3–4.4 представлен внешний вид форм программы в конструкторе форм.

Главное назначение формы `Default.aspx` — обеспечение навигации между страницами приложения. Ссылки на формы приложения можно обеспечить с помощью компонента `LinkButton`. Обратите внимание на свойство `PostBaskUrl` этих компонентов, так как именно оно содержит адрес ссылки на другую страницу (таблица 4.1). Символ «~» впереди адреса страниц указывает на то, что страницы размещены в папке проекта. В противном случае надо указывать полный путь в виде `URL`.

Форма (страница) Buyers.aspx обеспечивает роботу с таблицей БД buyers («Покупателе»). Ее внешний вид в конструкторе приведено на рисунке 4.4.

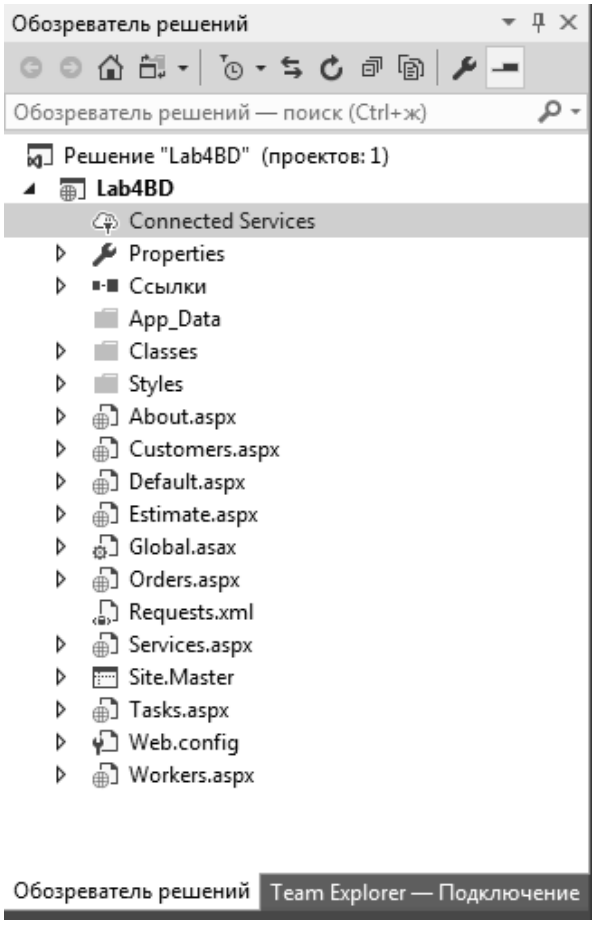

Рисунок 4.2 – Окно Solution Explorer (Обозреватель решений)

Таблица 4.1 — Значение свойств для компонентов `LinkButton` главной формы программы

Свойство		
Id	Text	PostBaskUrl
lbEstimate	Смета	~/Estimate.aspx
lbWorkers	Сотрудники	~/Workers.aspx
lbServices	Услуги	~/Services.aspx
lbCustomers	Заказчики	~/Customers.aspx
lb Orders	Заказы	~/Orders.aspx
lbTasks	Отчеты	~/Tasks.aspx

Рисунок 4.3 – Внешний вид формы `Default.aspx`

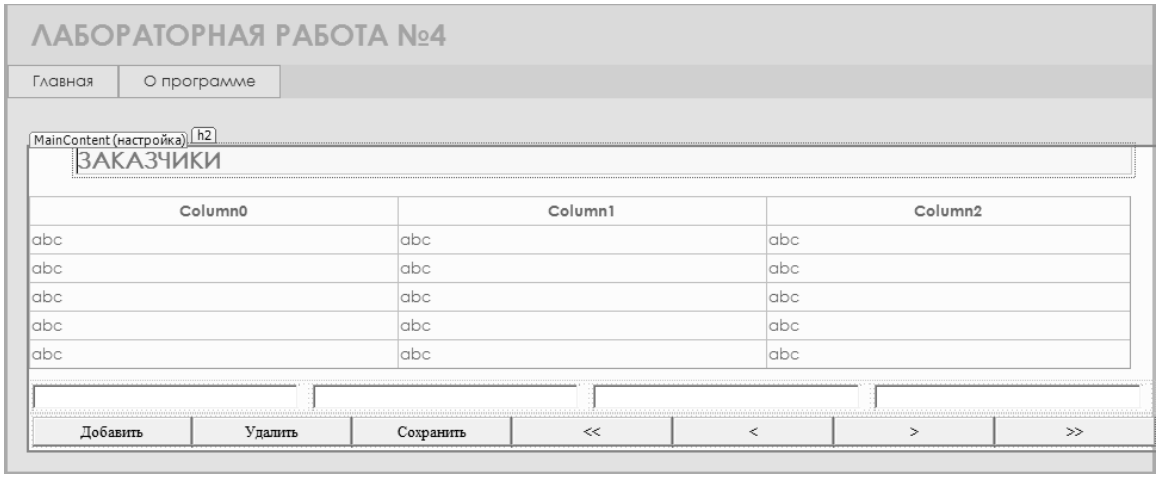

Рисунок 4.4 – Внешний вид формы `Customers.aspx`

Данные выводятся в компонент GridView с именем gvCustomers. Для добавления новых и редактирование существующих данных ниже, на панели pnNew установлены текстовые поля txtID, txtName, txtAddress и txtPassport.

Управление операциями осуществляется с помощью кнопок addButton (добавление), saveButton (завершение), deleteButton (удалить), а навигация по записям — с помощью кнопок toBegin (первый), toPrevious (предыдущий), toNext (следующий) и toLast (последний).

Для создания столбцов таблицы GridView использован конструктор, доступ к которому обеспечивает свойство Columns. Вы можете создавать столбцы, которые могут отображать текст, рисунки, содержать списки и флажки.

4.3.3 Программирование приложения

Разработка приложения состоит в написании программного кода форм (ASP.NET-страниц), выполняющего следующие операции:

– чтение данных из таблиц БД;
– добавление новых записей в таблице БД;
– редактирование (изменение) записей в таблицах БД;
– удаление записей из таблиц БД;
– навигацию по записям таблиц БД;
– формирование отчетов.

В программном коде ниже представлен текст страницы Customers.aspx:

```
using System;
using System.Data;
using MySql.Data.MySqlClient;

public partial class Customers : System.Web.UI.Page
{
  // соединение с БД
  private MySqlConnection msc;

  // адаптер доступа к данным
```

```
private MySqlDataAdapter adapter;

// набор данных
private DataSet dataSet;

// индекс выбранной строки
private static int selectedIndex = 0;

// количество записей в наборе данных
private int rowCount;

// метод для экранирования строковых данных
public String commas(String name)
{
 return "\"" + name + "\"";
}

// метод, используемый для добавления записи в таблицу
// keys - список полей
// values - знеачения полей
public void insertRow(String[] keys, String[] values)
{
  try {
    // формируем запрос insert (часть с перечнем полей)
    String query = "INSERT INTO customers (" + keys[0];
    for (int i = 1; i < keys.Length; i++)
      query += ", " + keys[i];

    // продолжаем формировать запрос insert
    // (часть с values)
    query += ") VALUES(" + values[0];
    for (int i = 1; i < values.Length; i++)
      query += ", " + values[i];
    query += ")";

    // выполняем запрос
    MySqlCommand cmd = new MySqlCommand(query, msc);
    cmd.ExecuteNonQuery();
  }
   catch (MySqlException) { }
}
```

```
// метод, используемый для обновления записи в таблице
// keys - список полей
// values - знеачения полей
// idName - имя первичного ключа
// value - значение первичного ключа
public void updateRow(
 String[] keys, String[] values, String idName, int value)
{
 try {
   // формирование запроса update
   String query = "UPDATE customers SET " +
     keys[0] + " = " + values[0];

   // продолжение формирования запроса update
   for (int i = 1; i < keys.Length; i++)
     query += ", " + keys[i] + " = " + values[i];

   // формирование условия обновления записи
   query += " WHERE " + idName + " = " + value;

   // выполнение запроса
   MySqlCommand cmd = new MySqlCommand(query, msc);
   cmd.ExecuteNonQuery();
 }
 catch (MySqlException) { }
}

// метод, используемый для удаления записи в таблице
// idName - имя первичного ключа
// value - значение первичного ключа
public void deleteRow(String idName, String value) {
  try {
    // формирование и выполнение запроса удаления записи
    MySqlCommand cmd = new MySqlCommand("DELETE FROM" +
      " customers WHERE " + idName + " = " +
      value, msc);
    cmd.ExecuteNonQuery();
  }

  catch (MySqlException) { }
}
```

```
// обработчик события инициализация страницы
protected void Page_Init(object sender, EventArgs e)
{
   // вызываем метод инициализации страницы
   loadPage();
}

// метод инициализации страницы
public void loadPage()
{
   // создание и открытие соединения с БД
 msc = new MySqlConnection("server=localhost; " +
    " user='root'; database=lab4;" +
    " port=3306; password=''; CharSet=utf8;");
 msc.Open();

 // выполнение запроса выборки данных из таблицы customers
 using (MySqlCommand cmd =
    new MySqlCommand(" SELECT * FROM customers; ", msc))
 {
    adapter = new MySqlDataAdapter(cmd);
    dataSet = new DataSet();
    adapter.Fill(dataSet, "customers");
    gvCustomers.DataSource = dataSet.Tables[0];
    gvCustomers.DataBind();
 }

 // запоминаем кол-во записей
 rowCount = dataSet.Tables[0].Rows.Count;

 // вызываем метод установки значений
 // в поля для редактирования значений текущей записи
 setValue();
}

// метод, задающий значения в поля редактирования
// для текущей записи
public void setValue()
{
 txtID.Text =
   dataSet.Tables[0].Rows[selectedIndex][0].ToString();
```

```
 txtName.Text =
   dataSet.Tables[0].Rows[selectedIndex][1].ToString();
 txtAddress.Text =
   dataSet.Tables[0].Rows[selectedIndex][2].ToString();
 txtPassport.Text =
   dataSet.Tables[0].Rows[selectedIndex][3].ToString();
}

// обработчик события Click от кнопки перехода
// на следующую запись
protected void toNext_Click(object sender, EventArgs e)
{
  if(selectedIndex < rowCount-1)
    selectedIndex++;
  setValue();
}

// обработчик события Click от кнопки перехода
// на предыдущую запись
protected void toPrevious_Click(
  object sender, EventArgs e)
{
  if (selectedIndex > 0)
    selectedIndex--;
  setValue();
}

// обработчик события Click от кнопки перехода
// на первую запись
protected void toBegin_Click(object sender, EventArgs e)
{
  selectedIndex=0;
  setValue();
}

// обработчик события Click от кнопки перехода на
// последнюю запись
protected void toLast_Click(object sender, EventArgs e)
{
  selectedIndex = rowCount-1;
  setValue();
```

```
}

// обработчик события Click нажатия на кнопку сохранить
protected void saveButton_Click(
   object sender, EventArgs e)
{
 // формируем массивы с именами полей и значениями
 String[] keys =
   new String[3] {
     “name_customer”, “address”, “passport”
   };

 String[] values =
   new String[3] {
     commas(txtName.Text),
     commas(txtAddress.Text),
     txtPassport.Text
   };

 if (txtID.Text == "") {
   // если поле txtID пустое, то запись нужно создать
   insertRow(keys, values);
   selectedIndex = 0;

   // перезагружаем страницу
   loadPage();
 }else{
   // иначе запись редактируетс
   // обновляем ее
   updateRow(
     keys, values, "id_customers",
     dataSet.Tables[0].Rows[selectedIndex][0].ToString()
   );
  selectedIndex = 0;

  // перезагружаем страницу
  loadPage();
 }
}

// обработчик события нажатия на кнопку добавить
```

```
  protected void addButton_Click(object sender, EventArgs e)
  {
     txtID.Text      = ""; txtName.Text     = "";
     txtAddress.Text = ""; txtPassport.Text = "";
     selectedIndex = 0;
  }

  // обаботчик события нажатия на кнопку удалить
  protected void deleteButton_Click(
    object sender, EventArgs e)
  {
     // удаляем запись
     deleteRow("id_customers",
       dataSet.Tables[0].Rows[selectedIndex][0].ToString());
   selectedIndex = 0;

   // перезагружаем страницу
   loadPage();
  }
}
```

4.4 Задание к лабораторной работе

Разработайте Web-приложение для управления БД, которая была создана Вами при выполнении 1-й и 2-й лабораторных работ.

Приложение должено содержать: главную страницу с гиперссылками на другие страницы (формы), страницы (Web-формы) для доступа к данным, страницы отчетов, которые вызываются из главной формы.

Доступ к справочным таблицам должен осуществляться через связанные элементы управления. Для доступа к основной таблице используйте элемент управления `DataGrid`. Для навигации по записям таблиц БД установите кнопки `Button`.

Используя результаты запроса с параметрами, создайте отчет. Отчет должен выводить на отдельной форме.

Запрограммируйте приложение. Обеспечьте условия для минимизации ошибок пользователя. Обеспечьте кеширование изменений. Создайте систему предупреждений для пользователя в случае возникновения ошибки.

4.5 Требования к отчету

Отчет оформляется на стандартных листах белой бумаги формата А4 (210х297 мм). Текст пишется с одной стороны листа чернила синего или черного цвета или печатается на принтере.

Бланк отчета готовится во время домашней подготовки к работе и должен содержать:

– данные о студенте: фамилия и инициалы, шифр группы;

– тему и цель работы;

– дату выполнения работы;

– набор данных, необходимых для выполнения работы: задача к работе, код программы, короткие теоретические сведения о функциях и библиотеках Visual Basic.NET, которые использовались при подготовке отчета.

После выполнения работы к отчету подшивается распечатка результатов.

Отчет должен быть выполнен аккуратно. Неаккуратно выполненные отчеты, а также ксерокопии чужих отчетов или их фрагментов к защите не допускаются.

4.6 Контрольные вопросы

1. Что такое привязка данных в ASP.NET?
2. Чтопредставляет собой простая привязка данных?
3. Как осуществить простую привязку данных?
4. Чтопредставляет собой сложная привязка данных?
5. Как осуществить сложную привязку данных?
6. Как осуществить привязку данных с помощью DataReader?
7. Для чего используется кеширование данных в ASP.NET и какие его типы существуют?
8. Добавление данных в таблицу с помощью `DataTable`, приведите пример.
9. Добавление данных в таблицу с помощью `MySqlCommand`, приведите пример.
10. Обновление строки в таблице с ипользованием `DataRow`, приведите пример.

11. Обновление строки в таблице с ипользованием `MySqlCommand`, приведите пример.

12. Какие Вы знаете способы удаления строки из набора данных? Перечислите их, приведите примеры.

13. Основное назначение транзакции в ADO.NET?

14. Как реализовать транзакцию в программе на C#?

15. Каким образом можно осуществить перемещение по строкам в компоненте GridView?

5 ЛАБОРАТОРНАЯ РАБОТА № 5

Тема: Работа с базами данных в Java.

Цель работы: научиться работать с базами данных с использованием технологии JDBC и языка программирования JAVA.

5.1 Общие сведения о работе с базами данных в Java

JDBC — это прикладной программный интерфейс (далее API) Java для работы с базами данных через SQL-запросы. Он состоит из множества классов и интерфейсов, написанных на языке Java.

С помощью JDBC можно выполнять SQL-запросы почти ко всем реляционным БД. JDBC расширяет и без того богатую функциональность Java. Например, можно опубликовать в Интернет HTML-страницу, содержащую сервлет, связанный с БД на сервере. Еще один пример: организация с помощью JDBC может подключить всех сотрудников к одной БД, даже если мы имеем дело с конгломератом операционных систем на рабочих станциях сотрудников - Windows, Macintosh, UNIX.

JDBC позволяет: устанавливать соединение с БД, отсылать SQL-запросы и обрабатывать их результаты, используя двух- и трехзвенные модели для доступа к БД.

В двухзвенной модели приложение (апплет, сервлет) на языке Java обращается непосредственно к БД, используя JDBC-драйвер. SQL-запросы отсылаются к СУБД, а результаты – обратно к пользователю. БД может находиться на другой машине, с которой пользователь соединяется по сети. И пользовательская машина, и сервер БД работают в конфигурации клиент/сервер. В качестве сети может выступать Intranet, соединяя между собой работников корпорации, или Internet.

В трехзвенной модели команды поступают в сервис среднего звена, который отсылает SQL-выражения к СУБД. СУБД обрабатывает SQL, отсылая запросы в указанный сервис, который затем возвращает результат конечному пользователю.

Трехзвенная модель привлекательна тем, что может контролировать доступ и изменения, вносимые в корпоративную БД. Другое достоинство модели заключается в том, что программист может реализовать

свой собственный предметно-ориентированный API, который обеспечивает трансляцию средним звеном низкоуровневых SQL-запросов. Кроме того, во многих случаях трехзвенная архитектура может увеличить производительность.

Вплоть до недавнего времени промежуточный слой было принято создавать на таких высокопроизводительных языках как C или C++. С созданием компиляторов из байт-кода языка Java в эффективный машинный код реализация промежуточного звена на Java становится практически выгодной. Достоинства языка Java, которые в этом случае переносятся на промежуточное звено, - это гибкость, многопоточность и защищенность. JDBC в этой схеме предоставляет доступ к БД из промежуточного звена, написанного на Java.

Ведущие производители баз данных, ПО промежуточного уровня и инструментальных средств встраивают поддержку JDBC в свои новые продукты. Это означает, что заказчики могут создавать приложения Java, имея свободу выбора из множества конкурирующих между собой продуктов, и подбирать решение под свои конкретные особенности.

5.2 Управление данными с помощью Java API

5.2.1 Создание соединения с базой данных

Пакет JDBC предназначен для работы с диспетчерами СУБД от различных разработчиков, поэтому для подключения к базе данных среда выполнения Java должна загрузить драйвер для указанной базы данных.

Драйверы JDBC обычно создаются поставщиками СУБД. Главная задача лрайвера заключается в обработке JDBC-подключений и команд, поступающих от JAVA-приложения, и в генерации машинно-зависимых вызовов к базе данных.

Загрузка драйвера осуществляется следующим образом:

```
Class.forName(driverName),
```

где `driverName` – имя загружаемого драйвера.

Например, для загрузки драйвера для работы с СУБД mySQL следует написать:

```
Class.forName(“org.gjt.mm.mysql.Driver”).
```

Этот драйвер не входит в стандартную поставку JDK, его следует загрузить дополнительно.

Существует особый вид драйвера — мост JDBC-ODBC. Это драйвер JDBC, реализующий операции JDBC путем трансляции их в операции ODBC.

ODBC — промышленный стандарт, который обеспечивает доступ к различным системами управления базами данных, обеспечивая при этом максимально возможную совместимость. Технология ODBC используется в Windows для подключения клиента к удаленной базе данных. Средства ODBC находятся на компьютере клиента и каждый источник данных выглядит для приложения одинаково.

Схема архитектуры ODBC показана на рисунке 5.1.

Операции JDBC с точки зрения ODBC — это обычное приложение. Мост JDBC-ODBC таким образом предоставляет JDBC-интерфейс к любым СУБД, для которых доступен ODBC-драйвер. Загрузка драйвера моста осуществляется следующим образом:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").
```

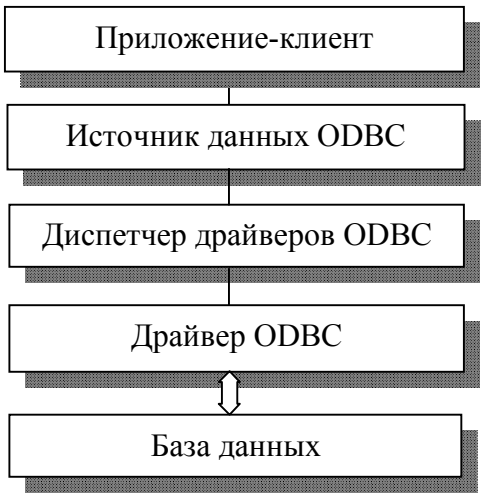

Рисунок 5.1 – Схема архитектуры ODBC

После загрузки драйвера можно устанавливать соединение с базой данных. Это реализуется с помощью метода `DriverManager.getConnection`, который имеет следующий синтаксис:

```
Connection conn = DriverManager.getConnection(
                                        url, user, password);
```

где conn – имя устанавливаемого соединения;

user – имя пользователя базы данных;

password – пароль пользователя;

url – указание адреса базы данных и определенного драйвера, который устанавливает соединение с данной базой данных. Он состоит из трех частей:

```
jdbc:<subprotocol>:<subname>
```

где <subprotocol> – имя драйвера или механизма соединения с базой данных;

<subname> – идентификатор базы данных.

Например, при использовании JDBC-ODBC-моста строка соединения может иметь вид:

```
Connection conn = DriverManager.getConnection(
    "jdbc:odbc:myDB", "admin", "password");
```

В данном случае myDB — это системный DSN, определенный в диспетчере источников данных ODBC. DSN (Data Source Name) — имя источника данных, которое содержит информацию о драйвере СУБД, адресе базы данных, учетной записи и пароле. Это дает возможность упростить приложению клиента процесс установления соединения с базой данных. Подробно создание DSN рассмотрено в [3].

Если же база данных удаленная (например, находится в Internet), то в состав URL должен быть включен сетевой адрес, подчиняющийся следующим соглашениям:

```
//hostname:port/subsubname
```

Пусть "dbNet" – это протокол соединения к хосту в Internet. Тогда URL может выглядеть так:

```
jdbc:dbNet://wombat:356/MyDB.
```

Для СУБД mySQL соединение может быть установлено следующим образом:

```
Connection conn = DriverManager.getConnection(
  "jdbc:mysql://localhost:3306/dbName", "admin", password");
```

Получить информацию об установленном соединении можно с помощью метода `getMetaData()`:

```
DatabaseMetaData dma = conn.getMetaData();
```

Класс `DatabaseMetaData` изучите самостоятельно.

5.2.2 Создание SQL-выражений и их выполнение

Для выполнения SQL-запросов к БД используется объект `Statement`. Существует три типа объектов `Statement`, которые служат контейнерами для выполнения SQL-выражений через данное соединение:

– `Statement`;

– `PreparedStatement` (наследник `Statement`);

– `CallableStatement` (наследник `PreparedStatement`).

Они специализируются на различных типах запросов: `Statement` используется для выполнения простых SQL-запросов без параметров; `PreparedStatement` — для выполнения прекомпилированных SQL-запросов с или без входных параметров; `CallableStatement` – для вызовов хранимых процедур.

5.2.2.1 Обычные запросы

Как только соединение с БД установлено, его можно использовать для выполнения SQL-запросов. Объект Statement создается методом `Connection.createStatement`, как показано ниже:

```
Connection conn =
  DriverManager.getConnection (
```

```
    "jdbc:odbc:myDB", "sunny", "");
Statement stmt = con.createStatement();
```

Посылаемое в БД SQL-выражение передается в качестве аргумента одному из методов объекта `Statement` для выполнения SQL-запроса:

```
ResultSet rs =
  stmt.executeQuery("SELECT a, b, c FROM Table");
```

Интерфейс `Statement` предоставляет три различных метода выполнения SQL-выражений: `executeQuery`, `executeUpdate` и `execute`, в зависимости от SQL-запроса.

Метод `executeQuery` необходим для запросов, результатом которых является единственный набор значений (например, запросов SELECT).

Метод `executeUpdate` используется для выполнения операторов `INSERT`, `UPDATE` или `DELETE`, а также для операторов DDL (Data Definition Language - язык определения данных), например, `CREATE TABLE` и `DROP TABLE`. Результатом оператора `INSERT`, `UPDATE` или `DELETE` является модификация одного или более полей в некотором количестве строк таблицы. Метод `executeUpdate` возвращает целое число, показывающее, сколько строк было модифицировано. Для выражений типа `CREATE TABLE` и `DROP TABLE`, которые не оперируют над строками, возвращаемое методом `executeUpdate` значение всегда равно нулю.

Метод `execute` используется, когда операторы SQL возвращают более одного набора данных, более одного счетчика обновлений или и то, и другое.

Все методы выполнения SQL-запросов закрывают предыдущий набор результатов (result set) для текущего объекта `Statement`. Это означает, что перед тем как выполнять следующий запрос над тем же объектом `Statement`, надо завершить обработку результатов предыдущего (`ResultSet`).

Объекты Statement закрываются автоматически с помощью сборщика мусора виртуальной машины Java. Тем не менее рекомендуется закрывать их явно после того, как в них отпадает необходимость. Закрытие объектов Statement освобождает ресурсы СУБД и позволяет избежать проблем с памятью.

5.2.2.2 Запросы с параметрами

Как уже упоминалось в предыдущем разделе, для выполнения запросов с параметрами используется интерфейс PreparedStatement. Этот интерфейс является наследником Statement и отличается от него следующим:

– экземпляры PreparedStatement сохраняют скомпилированные SQL-выражения. Именно поэтому они называются prepared (подготовленные);

– SQL-выражения в PreparedStatement могут иметь один или более входных параметров.

Входной параметр - это параметр, значение которого не указывается при создании SQL-выражения. Вместо него в выражении на месте каждого входного параметра ставится знак «?». Значение вместо каждого вопросительного знака устанавливается методами setXXX перед выполнением запроса.

Поскольку объекты PreparedStatement скомпилированны, исполнение этих запросов может происходить несколько быстрее, чем в объектах Statement. В результате SQL-выражения, которые исполняются часто, в целях улучшения производительности создают в виде объектов PreparedStatement.

Оставаясь подклассом класса Statement, класс PreparedStatement наследует все функции от Statement. В дополнении к ним он добавляет методы установки входных параметров. Кроме того, три метода — execute, executeQuery и executeUpdate – модифицированы таким образом, что не имеют аргументов. Старые методы класса Statement (которые принимают SQL-выражения в качестве единственного аргумента) не должны использоваться в объекте PreparedStatement.

Следующий фрагмент кода, где conn - это объект класса Connection, создает объект PreparedStatement, содержащий SQL-выражение с двумя параметрами:

```
PreparedStatement pstmt =
  conn.prepareStatement (
    "UPDATE table4 SET m = ? WHERE x = ?");
```

Объект pstmt содержит выражение "UPDATE table4 SET m = ? WHERE x = ? ", которое уже отослано в СУБД и подготовлено для выполнения.

Перед выполнением объекта PreparedStatement надо установить значения всех его параметров. Это делается с помощью методов setXXX, где XXX - это тип параметра. Например, если параметр имеет Java-тип long, следует использовать setLong. Первый аргумент методов setXXX –это порядковый номер параметра, а второй — значение, в которое надо его установить. Например, следующий код устанавливает первый параметр в значение 123, а второй — в 100:

```
pstmt.setLong(1, 123);
pstmt.setLong(2, 100);
```

После установки параметра его можно использовать при многократном выполнении выражения до тех пор, пока не будет вызван метод clearParameters, который предназначен для очистки значений параметров.

Один и тот же объект PreparedStatement может выполняться много раз, если нижестоящий драйвер или СУБД будут сохранять выражение (statement) в открытом состоянии даже после того как произойдет фиксация. Иначе не имеет смысла пытаться улучшить производительность заменой Statement на PreparedStatement.

Приведем пример установки параметров для объекта pstmt, созданного в предыдущем примере:

```
pstmt.setString(1, “Hi”);
for (int i = 0; i < 10; i++) {
```

```
  pstmt.setInt(2, i);
  int rowCount = pstmt.executeUpdate();
}
```

Данная операция устанавливает значения обоих параметров и выполняет `pstmt` 10 раз. Как уже было отмечено, программа не должна закрывать `pstmt`. В этом примере первый параметр устанавливается в «Hi» и остается неизменным. Второй параметр устанавливается в последовательные целые значения, начиная от 0 и заканчивая 9.

5.2.2.3 Получение результирующего набора

Объект `ResultSet` содержит все записи, удовлетворяющие условиям в SQL-выражении и предоставляет доступ к данным в этих строках посредством набора get-методов, которые организуют доступ к полям текущей записи. Метод `ResultSet.next` используется для перемещения к следующей записи `ResultSet`, делая ее текущей.

Набор данных результата является таблицей с заголовками полей и соответствующих значений, возвращенных запросом. Например, если мы имеем запрос `SELECT a, b, c FROM Table1`, то набор результата будет в следующей форме:

a	b	C
12345	Cupertino	CA
83472	Redmond	WA
83492	Boston	MA

Следующий фрагмент кода демонстрирует выполнение SQL-запроса, который возвращает коллекцию записей, в которой поле 1 содержит данные типа `int`, поле 2 — `String` и поле 3 — массив байтов:

```
java.sql.Statement stmt = conn.createStatement();
ResultSet r =
   stmt.executeQuery("SELECT a, b, c FROM Table1");
while (r.next())
{
  // Напечатать значения в текущей строке.
  int i = r.getInt("a");
```

```
  String s = r.getString("b");
  float f = r.getFloat("c");
  System.out.println("ROW = " + i + " " + s + " " + f);
}
```

Объект `ResultSet` содержит так называемый курсор, который указывает на текущую строку данных. Каждый раз, когда выполняется метод `next`, курсор перемещается на одну запись вниз. Изначально курсор позиционирован перед первой строкой, и первый вызов `next` перемещает его на первую строку (она становится текущей). С каждым успешным вызовом `next` курсор перемещается вниз на одну строку, начиная с самой верхней в `ResultSet`.

Курсор сохраняется до тех пор, пока не закроется объект `ResultSet` или его родительский объект `Statement`.

Методы `getXXX` предоставляют доступ к значениям в полях в текущей записи. В пределах одной записи значения могут быть считаны в любом порядке, но ради обеспечения большей совместимости рекомендуется считывать их подряд слева направо и делать это только один раз.

Для указания поля можно использовать либо его имя, либо его номер. Например, если второе поле объекта `ResultSet rs` называется «`title`» и хранит строковое значение, то извлечь его можно одним из двух способов:

```
String s = rs.getString("title");
String s = rs.getString(2);
```

Поля нумеруются слева направо, начиная с 1. Имена полей в вызове методов `getXXX` нечувствительны к регистру букв.

5.2.2.4 Обновление, вставка и удаление данных из таблиц

Обновление, вставка и удаление данных из таблиц осуществляется посредством объекта `Statement` (см. раздел 3.2). Запросы на указанные действия могут быть как простыми, так и с параметрами.

Рассмотрим примеры осуществления указанных действий с базой данных.

Пусть в базе данных торгового предприятия есть 3 таблицы:

– *Справочник товаров* (КодТовара, Наименование, ЕдИзм, Цена);

– *Справочник покупателей* (КодПокупателя, Наименование, Телефон);

– *Продажи* (Дата, КодПокупателя, КодТовара, Количество).

Допустим, цены на одну из групп товаров повысились. Принимая во внимание, что код продукции содержит в себе и код группы, к которой он относится, то для обновления цен необходимо использовать следующий запрос с параметрами:

```
Connection conn = DriverManager.getConnection(
     "jdbc:odbc:myDB", "admin", "");
PreparedStatement pstmt = con.createStatement(
  "UPDATE Товары SET Цена = Цена * ? " +
  " WHERE КодТовара Like '?*'");
pstmt.setLong(1, 1.1);
pstmt.setLong(2, 2);
rowCount=pstmt.executeUpdate();
```

В результате выполнения этого запроса в таблице «Товары» на 10% повысятся цены на товары группы 2.

Этот же запрос можно применить для изменения цен с другими параметрами. Например, снижение цены на 20% товаров группы 1 осуществляется следующим образом:

```
pstmt.setLong(1, 0.8);
pstmt.setLong(2, 1);
rowCount=pstmt.executeUpdate();
```

При ликвидации одного из клиентов как юридического лица нужно удалить запись о нем из справочника покупателей. Удалим запись о покупателе ООО «Интер»:

```
Connection conn = DriverManager.getConnection(
  "jdbc:odbc:myDB", "admin", "");
Statement stmt = con.createStatement();
integer RecordCount = stmt.executeUpdate(
  "DELETE Покупатели.Наименование " +
  " FROM Покупатели WHERE " +
  " Покупатели.Наименование='ООО Интер'"
```

```
);
```

Вставка записей в таблицу базы данных осуществляется аналогичным образом.

5.2.3 Транзакции в JDBC

Транзакция – группа команд, которая может быть зафиксирована после успешного выполнения всех команд, либо отвергнута (выполнен откат), если при выполнении хотя бы одной из команд произойдет какая-то ошибка.

Основная причина использования транзакций заключается в поддержании целостности базы данных. Предположим, например, что в базу данных требуется вставить данные о новой книге. В этом случае нужно одновременно обновить данные в таблицах Книги, Авторы и АвторыКниг. Если данные будут правильно обновлены только в первых двух таблицах, то в таком случае будут получены противоречивые данные о книгах и авторах.

Транзакцию следует зафиксировать (committed), если успешно выполнены все команды транзакции. В противном случае выполняется откат (rollback), то есть отменяются все изменения в базе данных, которые выполнялись после предыдущей зафиксированной транзакции. Запросы могут выполняться только по отношению к зафиксированным данным!

По умолчанию соединение с базой данных обладает возможностью автоматической фиксации (autocommit mode), то есть каждая SQL-команда фиксируется после ее успешного выполнения. Причем нельзя выполнить откат для зафиксированной команды.

Для проверки текущего режима автоматической фиксации нужно вызвать метод `getAutoCommit` класса `Connection`. Для отключения режима автоматической фиксации нужно вызвать показанный ниже метод:

```
conn.setAutoCommit(false);
```

После этого можно приступать к созданию объекта `Statement`.

```
Statement stat = conn.createStatement();
```

Затем нужное количество раз вызывается метод `executeUpdate`:

```
stat.executeUpdate(commandl);
stat.executeUpdate(command2);
```

Здесь `command1`, ..., `commandN` – SQL-запросы на выполнение действий с базой данных.

После выполнения всех этих команд необходимо вызвать метод `commit()`.

```
conn.commit();
```

В случае возникновения какой-либо ошибки нужно выполнить откат всех предыдущих команд (кроме `commit`) с помощью метода `rollback`:

```
conn.rollback();
```

Откат обычно используется, если выполнение транзакции прерывается исключительной ситуацией `SQLException`.

5.2.4 Исключения класса SQLException

Класс `SQLException` является наследником класса `java.lang.Exception` (см. рис. 3.1).

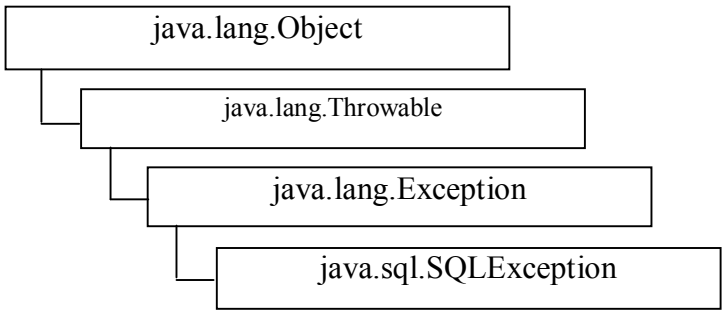

Рисунок 5.1 – Иерархия объектов

Исключение класса SQLException содержит информацию об ошибке доступа к базе данных.

Каждое исключение содержит такую информацию:

– текстовое описание ошибки;

– строку «SQLstate», которая соответствует соглашению XOPEN SQLstate;

– код ошибки (тип INTEGER), которые специфичны для каждого поставщика. Обычно это код ошибки, возвращаемой базой данных;

– ссылка на следующее исключение, которая может быть использована для получения дополнительной информации.

Конструктор класса SQLException может иметь следующий вид:

```
SQLException();
SQLException(String Reason);
SQLException(String Reason, String SQLstate);
SQLException(String Reason, String SQLstate, int vendor-
Code);
```

где Reason – описание исключения;

SQLstate – XOPEN-код, идентифицирующий исключение;

vendorCode – код исключения, определенный поставщиком базы данных.

Класс SQLException содержит следующие методы:

– getErrorCode() – извлекает определенный поставщиком код исключения для данного объекта SQLException;

– getNextException() – извлекает исключение, следующее за текущим;

– getSQLstate() – извлекает SQLstate для данного объекта SQLException;

– setNextException(SQLException ex) – добавляет объект SQLException в конец списка.

Текстовое описание исключения можно получить с помощью метода, унаследованного от класса Throwable getMessage().

5.3 Пример использования JDBC

Для примера рассмотрим простое приложение, позволяющее просматривать и редактировать данные из двух таблиц estimate и customers (структура БД идентична рассмотренной ранее в лабораторной работе № 4).

В состав приложения будут входить следующие классы:

– `Driver` — обеспечивает подключение к базеданных;

– `WorkDataBase` — обеспечивает основные функции по работе с БД;

– `Frame` — обеспечивает отображание данных.

Исходный код класса `Driver` приведен ниже:

```
package db;
import java.sql.*;
import static java.lang.System.out;

// класс, обеспечивающий подключение к БД
public class Driver {
  private static Connection connection;

  // конструктор
  public Driver(){
    // попытка создать подключение к БД
    try {
        connection = DriverManager.getConnection(
           “jdbc:mysql://localhost:3306/lab5”, “root”, “”);
        out.println("Подключение к БД установлено");
    }catch(SQLException  e) {
        out.println("Не удалось установить подключение");
    }
  }

  // метод для получения ссылки на подключение
  public Connection getConnection() {
    return connection;
  }

  // метод закрытия подключения к БД
  public void closeConnection() {
```

```
    try {
      connection.close();
    } catch (SQLException e) {
      out.println("Не удалось закрыть соединение");
    }
  }
}
```

Исходный код класса `WorkDataBase` приведен ниже:

```
package db;

import java.sql.*;
import java.util.ArrayList;
import javax.swing.JComboBox;

public class WorkDataBase {
  // соединение с БД
  private Connection connection;

  // запросы для работы с БД
  private String requests[] = {
    // запрос для отбора данных из таблицы смета
    "SELECT " +
    "  estimate.id_estimate, orders.date_begin, " +
    "  services.name_service, workers.FIO " +
    "FROM" +
    "  estimate " +
    "  INNER JOIN orders" +
    "    ON estimate.id_order = orders.id_order" +
    "  INNER JOIN services" +
    "    ON estimate.id_service = services.id_service" +
    "  INNER JOIN workers" +
    "    ON estimate.id_worker = workers.id_worker " +
    "ORDER BY " +
    "  estimate.id_estimate; ",

    // запрос для отбора данных из таблицы заказчиков
    "SELECT " +
    "  id_customers, name_customer, address, passport " +
    "FROM " +
    "  customers c " +
    "ORDER BY " +
```

```
  "  id_customers;"
};

// заголовки
private Object header[] = null;

// данные
private Object data[][] = null;

// заголовки таблицы
private String headerTable[] = null;

// имена таблиц
private String tableName[] = {"estimate", "customers"};

// имена таблиц для отображения в интерфейсе
private String viewTableName[] = {"Смета", "Заказчики"};

// индекс таблицы, выбранный по умолчанию
private int selectTable = 0;

// конструктор класса
public WorkDataBase(Driver driver) {
  // запоминаем ссылку на соединение с базой
  connection = driver.getConnection();
}

// метод для заполнения данными  JComboBox
public void setComboBox(JComboBox comboBox) {
  for(String s: viewTableName)
    comboBox.addItem(s);
}

// заполнить значения для таблицы в зависимости от
// выбранной таблицы в comboBox
// select - индекс таблицы
public void fillTable(int select) {
  selectTable = select;
  try {
    // создание запроса
    Statement st = connection.createStatement();
```

```
 ResultSet rs = null;

// подготовка к выполнению запросов
 switch (select) {
 case 0:
   // заполняем данные для заголовков таблицы смета
   header =
     new Object[]{“Номер”, “Дата заказа”,
                  “Услуга”, “Сотрудник”};
   headerTable =
      new String[] {“id_order”, “id_service”,
        “id_worker”};
   break;
 case 1:
   // заполняем данные для заголовков таблицы заказчики
   header =
     new Object[]{“Код заказчика”,
       “Название заказчика”, “Адрес”, “Паспорт”};
   headerTable =
     new String[] {“name_customer”,
       “address”, “passport”};
   break;
 }

 // выполнение запроса
 rs = st.executeQuery(requests[select]);

 // перемещаемся на последнюю запись
 // и получаем кол-во записей
 rs.last();
 int w = rs.getRow();

 // определвем размер массива для хранения записей
 data = new Object[w][header.length];

 // заполняем массив данными
 rs.beforeFirst();
 int i = 0;

 // цикл по трокам
 while(rs.next()) {
```

```
        // цикл по столбцам
        for(int j=1;j<header.length+1;j++) {
          data[i][j-1] = rs.getString(j);
        }
        i++;
      }
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }


  // метод, получающий значения указанного столбца
  // из указанной таблицы
  public ArrayList<String> getColumnFromTable(String column-
Name, String tableName) {
    ArrayList<String> array = new ArrayList<String>();
    try {
      // выполняем запрос
      Statement st = connection.createStatement();
      ResultSet rs =
        st.executeQuery("SELECT " + columnName + " FROM " +
          tableName);

      // заполняем результат
      while(rs.next())
        array.add(rs.getString(1));
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return array;
  }

  // метод, получающий значение указанного столбца
  // из указанной таблицы по id
  public int getElement(String pk, String columnName, String
table, String pkName) {
    int p = 0;
    int id = Integer.parseInt(pk);

    try {
```

```
      Statement st = connection.createStatement();
      ResultSet rs = st.executeQuery("SELECT " +
        columnName + " FROM " + table + " WHERE " +
        pkName + " = " + id);
      rs.next();
      p = rs.getInt(1);
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return p;
  }

  // метод, выполняеющий вставку записи в таблицу
  // с указанными значениями
  // values - значения
  public void insertQuery(String values[]) throws SQLExcep-
tion {
    try {
      // формирование запроса
      String query =
        "INSERT INTO " + tableName[selectTable] + "(" +
         headerTable[0];

      for (int i = 1; i < headerTable.length; i++)
        query += ", " + headerTable[i];

      query += ") VALUES(" + values[0];

      for (int i = 1; i < values.length; i++)
        query += ", " + values[i];

      query += ")";

      // выполнение запроса
      Statement st = connection.createStatement();
      st.executeUpdate(query);
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }
```

```
  // метод, выполняеющий обновление записи в таблице
  // values - значения
  // idName - имя ключевого поля
  // value - значение ключевого поля
  public void updateQuery(String values[], String idName,
String value) throws SQLException {
    try {

      // формирование запросf
      String query = "UPDATE " + tableName[selectTable] +
        " SET " + headerTable[0] + " = " + values[0];

      for (int i = 1; i < headerTable.length; i++)
        query += ", " + headerTable[i] + " = " + values[i];

      query += " WHERE " + idName + " = " + value;

      // выполнение запроса
      Statement st = connection.createStatement();
      st.executeUpdate(query);
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }

  // метод, выполняеющий удаление записи из таблицы
  public void deleteQuery(String idName, String value)
throws SQLException {
    try {
      String query = "DELETE FROM " +
        tableName[selectTable] + " WHERE " + idName +
        " = " + value;
      Statement st = connection.createStatement();
      st.executeUpdate(query);
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }

  // метод, возвращающий заголовки
  public Object[] getHeader() {
```

```
    return header;
  }

  // метод, возвращающий данные
  public Object[][] getData() {
    return data;
  }
}
```

Для отображения на экране данных с помощью классов, приведенных выше, разработан класс `Frame`, представляющий собой форму, на которой размещены следующие элементы:

- `tableBox` — элемент управления `JComboBox`, используемый для выбора редактируемой таблицы;
- `table` — элемент управления `JTable`, используемый для отображения данных в табличной форме;
- `mainPanel` — элемент управления `JPanel`, панель на которойразмещаются основные компонеты;
- `saveButton` — элемент управления `JButton`, кнопка «Сохранить»;
- `addButton` — элемент управления `JButton`, кнопка «Добавить»;
- `deleteButton` — элемент управления `JButton`, кнопка «Удалить»;
- `panelForFields` — элемент управления `JPanel`, панель на котором будут размещаться элементы управления для редактирования активной записи;

Внешний вид формы приведен на рисунке 5.2

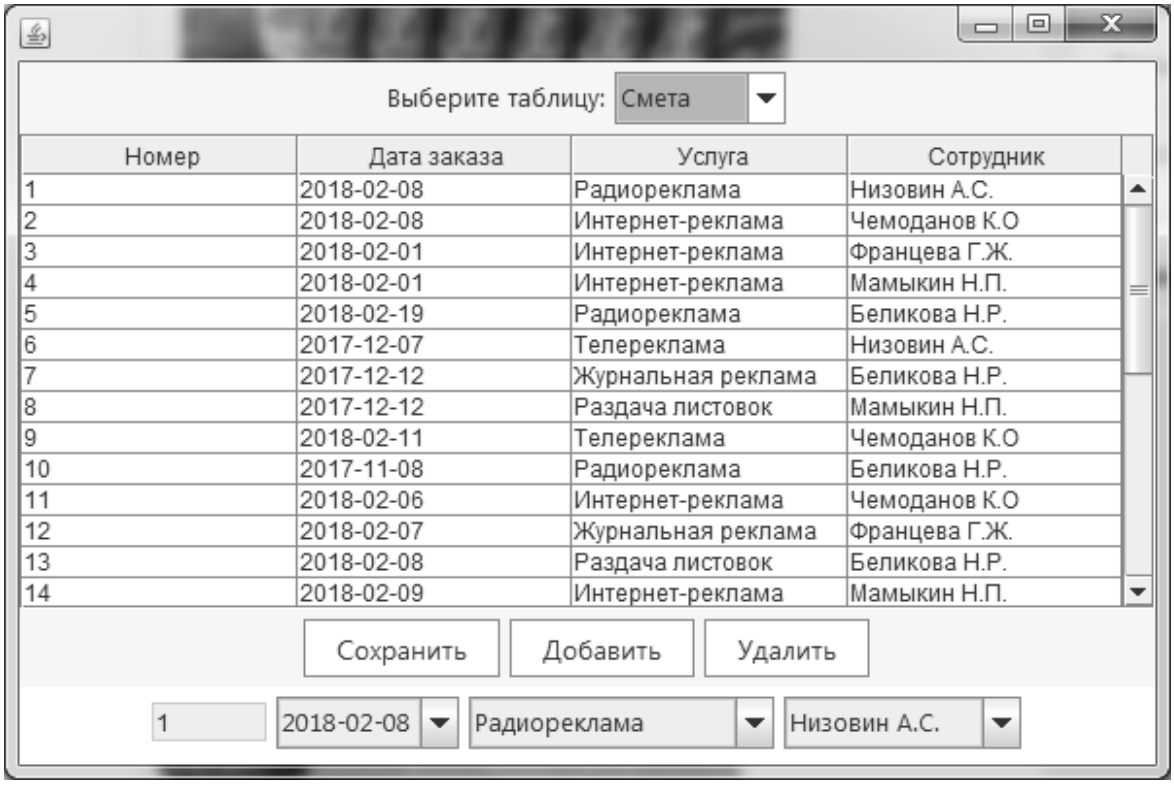

Рисунок 5.2 — Интерфейс разаработанного приложения

Исходный код класса `Frame` приведен ниже (часть автоматически сгенерированного кода средой разработки отсутствует):

```
package gui;

import java.awt.Dimension;
import java.awt.EventQueue;
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.event.*;
import java.sql.SQLException;

import db.Driver;
import db.WorkDataBase;

public class Frame {

  // фрейм - форма
  private JFrame frame;
```

```
// драйвер доступа к БД класс Driver
private Driver driver;

// база данных WorkDataBase
private WorkDataBase db;

// раскрывающийся список для перечня таблиц
private JComboBox tableBox;

// таблица для отображения данных
private JTable table;

// главная панель
private JPanel mainPanel;

// кнопки манипулировани яданными
private JButton saveButton;
private JButton addButton;
private JButton deleteButton;

// панель для полей редактирования записи
private JPanel panelForFields;
// элементы управления для формирования
// полей редактирования активной записи
private JTextField field1 = new JTextField();
private JTextField field2 = new JTextField();
private JTextField field3 = new JTextField();
private JTextField field4 = new JTextField();
private JTextField field5 = new JTextField();
private JTextField field6 = new JTextField();
private JComboBox box1 = new JComboBox();
private JComboBox box2 = new JComboBox();
private JComboBox box3 = new JComboBox();

// метод mai для запуска приложения
public static void main(String[] args) {
  EventQueue.invokeLater(new Runnable() {
    public void run() {
      try {
        Frame window = new Frame();
```

```
            window.frame.setVisible(true);
          } catch (Exception e) {
            e.printStackTrace();
          }
        }
      });
  }

  // конструктор класса
    public Frame() {
      // создание экземпляра класса Driver
      driver = new Driver();

      // инициализация элементов управления
      initialize();

      // создание экземпляра класса WorkDataBase
      db = new WorkDataBase(driver);

      // задание списка таблиц
      db.setComboBox(tableBox);
  }

  // метод задания поля для редактирования
  private void setField(JTextField field, String parameter,
boolean editable) {
  field.setColumns(10);
  addFieldToPanel(field, parameter, editable);
}

  // метод задания поля для редактирования
  private void setField(JTextField field, String parameter,
boolean editable, int size) {
  field.setColumns(size);
  addFieldToPanel(field, parameter, editable);
}

  // метод добавления поля на панель
  private void addFieldToPanel(JTextField field, String pa-
rameter, boolean editable) {
  field.setText(parameter);
```

```
  field.setEditable(editable);
  panelForFields.add(field);
}

  // метод задания раскрывающегося списка
  private void setComboBox(JComboBox box, ArrayList<String>
parameters) {
  panelForFields.add(box);
  for (String s : parameters) {
     box.addItem(s);
  }
  box.setFont(new Font("Segoe UI", Font.PLAIN, 13));
}


  // метод загрузки компонентов на панель и установка
  // значений по умолчанию
  private void loadComponentsOnPanel() {
    panelForFields.removeAll();
    panelForFields.updateUI();
    box1.removeAllItems();
    box2.removeAllItems();
    box3.removeAllItems();
    switch (tableBox.getSelectedIndex()) {
      // выбрана таблица смета
      case 0:
        setField(
          field1, table.getValueAt(0, 0).toString(),
          false, 5
        );
        setComboBox(box1,
          db.getColumnFromTable("date_begin", "orders")
        );
       setComboBox(box2,
          db.getColumnFromTable("name_service", "services")
       );
       setComboBox(box3,
         db.getColumnFromTable("FIO", "workers")
       );
       box1.setSelectedIndex(
         db.getElement(
```

```
            field1.getText(),
          "id_order", "estimate", "id_estimate") - 1
        );
        box2.setSelectedIndex(
          db.getElement(field1.getText(),
            "id_service", "estimate", "id_estimate") - 1
        );
        box3.setSelectedIndex(
          db.getElement(
            field1.getText(), "id_worker", "estimate",
            "id_estimate") - 1
        );
        break;

      // выбрана таблица заказчики
      case 1:
        setField(
          field1, table.getValueAt(0, 0).toString(),
          false, 5
        );
        setField(field2, table.getValueAt(0, 1).toString(),
          true);
        setField(field3, table.getValueAt(0, 2).toString(),
          true);
        setField(field4, table.getValueAt(0, 3).toString(),
          true);
        break;
      }
  }

  // обработка события изменения выбранной таблицы
  private void selectedChanged() {
    table.getSelectionModel().addListSelectionListener(new
ListSelectionListener() {
      public void valueChanged(ListSelectionEvent event) {
        if (!event.getValueIsAdjusting()) {
          int selected = tableBox.getSelectedIndex();
        // определение вбранной таблицы
        switch (tableBox.getSelectedIndex()) {
          // выбрана таблица смета
          case 0:
```

```
 field1.setText(
   table.getValueAt(
     table.getSelectedRow(), 0).toString()
 );
 box1.setSelectedIndex(
   db.getElement(
     field1.getText(), "id_order", "estimate",
       "id_estimate") - 1
  );
  box2.setSelectedIndex(
    db.getElement(
      field1.getText(), "id_service", "estimate",
      "id_estimate") - 1
  );
  box3.setSelectedIndex(
    db.getElement(field1.getText(), "id_worker",
    "estimate", "id_estimate") - 1
  );
  break;

  // выбрана таблица заказчики
case 1:
  field1.setText(
    table.getValueAt(table.getSelectedRow(),
       0).toString()
  );
  field2.setText(
    table.getValueAt(table.getSelectedRow(),
      1).toString()
  );
  field3.setText(
    table.getValueAt(table.getSelectedRow(),
      2).toString()
  );
  field4.setText(
    table.getValueAt(table.getSelectedRow(),
      3).toString()
  );
  break;
}
```

```
        }
      }
    });
  }

  // метод обновления значений строк либо добавление новых
  private void insertUpdateTable() {
    String values[];
    try {
      // определяем для какой таблицы действие
      switch (tableBox.getSelectedIndex()) {
        case 0:
          values =
            new String[]{
              (box1.getSelectedIndex() + 1) + "",
              (box2.getSelectedIndex() + 1) + "",
              (box3.getSelectedIndex() + 1) + ""
            };
          //Если поле с id пустое..
          if (field1.getText().equals("")) {
            // добавление записи
              db.insertQuery(values);
          } else {
            // обновление записи
            db.updateQuery(
              values, "id_estimate", field1.getText()
            );
          }
          break;
        case 1:
          values =
            new String[]{
              commit(field2.getText()),
              commit(field3.getText()),
              field4.getText()
            };
            //Если поле с id пустое
            if (field1.getText().equals("")) {
              // добавление записи
              db.insertQuery(values);
            } else {
```

```
              // обновление записи
              db.updateQuery(
                values, "id_customers", field1.getText());
            }
            break;
     }
    } catch (SQLException e) {
      JOptionPane.showMessageDialog(
        null, e.toString(), "Error",
          JOptionPane.ERROR_MESSAGE);
    }
}

// метод экранирования строки кавычками
private String commit(String s) {
  return "'" + s + "'";
}

// метод обновления таблицы и события к ней
private void refreshTable() {
  db.fillTable(tableBox.getSelectedIndex());

  // модель для JTable
  TableModel dtm = new TableModel();

  // задать заголовки
  dtm.setColumnIdentifiers(db.getHeader());
  Object[][] a = db.getData();

  // заполнить строки
  for (int i = 0; i < a.length; i++) {
    dtm.addRow(a[i]);
  }
  table = new JTable(dtm);

  //обновить элементы
  mainPanel.removeAll();
  mainPanel.updateUI();
  mainPanel.add(new JScrollPane(table));
  loadComponentsOnPanel();
  selectedChanged();
```

```
  }

  // запрет редактирования в таблице
  class TableModel extends ja-
vax.swing.table.DefaultTableModel {
    public boolean isCellEditable(int row, int col) {
      return false;
    }
  }

  // событие на выбор таблицы в comboBox
  private void comboboxListener() {
    tableBox.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        refreshTable();
      }
    });
  }

  // событие на кнопку Добавить
  private void addButtonListener() {
    addButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent arg0) {
        field1.setText("");
        field2.setText("");
        field3.setText("");
        field4.setText("");
        field5.setText("");
        field6.setText("");
        if (box1.getItemCount() > 0) {
          box1.setSelectedIndex(0);
        }
        if (box2.getItemCount() > 0) {
          box2.setSelectedIndex(0);
        }
        if (box3.getItemCount() > 0) {
          box3.setSelectedIndex(0);
        }
      }
    });
  }
```

```
// событие на кнопку Удалить
private void deleteButtonListener() {
  deleteButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
      try {
        if (table.getSelectedRow() != -1) {
          switch (tableBox.getSelectedIndex()) {
            case 0:
              db.deleteQuery(
                "id_estimate", field1.getText()
              );
              break;
            case 1:
              db.deleteQuery(
                "id_customers", field1.getText()
              );
              break;
          }
        }
        refreshTable();
      } catch (SQLException e) {
        JOptionPane.showMessageDialog(
          null, e.toString(), "Error",
           JOptionPane.ERROR_MESSAGE
        );
      }
    }
  });
}

// событие на кнопку Сохранить
private void saveButtonListener() {
  saveButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
      insertUpdateTable();
      refreshTable();
    }
  });
}
```

5.4 Задание к лабораторной работе

Разработайте программу, которая обеспечивает работу с базой данных:

1. Извлечение данных с помощью запросов.
2. Занесение новых данных.
3. Редактирование данных.
4. Удаление записей.

5.5 Требования к отчету

Отчет оформляется на стандартных листах белой бумаги формата А4 (210х297 мм). Текст пишется с одной стороны листа чернилами синего или черного цвета или печатается на принтере.

Бланк отчета готовится во время домашней подготовки к работе и должен содержать:

данные о студенте: фамилия и инициалы, шифр группы;

тему и цель работы;

дату выполнения работы;

набор данных, необходимых для выполнения работы: задание к работе, код программы, краткие теоретические сведения о функциях и библиотеках Java, которые использовались при подготовке отчета.

После выполнения работы к отчету подшиваются: распечатка программного кода, распечатка результатов.

Отчет должен быть выполнен аккуратно. Неаккуратно выполненные отчеты, а также ксерокопии чужих отчетов или их фрагментов к защите не допускаются.

5.6 Контрольные вопросы

1. Что представляет собой JDBC и для чего он используется?
2. Какие модели доступа к базе данных использует JDBC?
3. Приведите схему архитектуры ODBC.
4. Как подключиться к серверу MySQL с использованием JDBC?
5. Как осуществ загрузку драйвера базы данных с использованием JDBC?
6. Какие сведения о БД можно плоучить с помощью объекта каласса `DatabaseMetaData?`

6. Приведите синтаксис создания объекта `Connection`?

7. Назначение класса `Statement`, его отличие от `PreparedStatement` и `CallableStatement`.

8. Назначение класса `PreparedStatement`, его отличие от `Statement` и `CallableStatement`.

9. Назначение класса `CallableStatement`, его отличие от `PreparedStatement` и `CallableStatement`.

10. Что является результатом выполнения метода `executeUpdate` объекта `Statement`?

11. Для выполнения каких операторов языка SQL используется метод `executeUpdate`?

12. Использование параметров совместно с `PreparedStatement`.

13. Какие методы используются для доступа к данным, содержащимся в объекте `ResultSet`?

14. Опишите методы объекта ResultSet для перемещения по набору данных.

15. Какую информацию содержит исключение SQLException?

6 РЕКОМЕНДОВАННАЯ ЛИТЕРАТУРА

1. MySQL: базовый курс / Шелдон Роберт, Джоффрей Мойе. — М. : ООО «И.Д. Вильямс», 2008. — 880с.

2. MySQL. Справочник по языку / Компания MySQL AB. — М. : Издательский дом «Вильямс», 2005. — 816с.

3. SQL: полное руководство / Дж. Грофф, П. Вайнберг. — К. : Издательская группа BHV, 2001. — 816с.

4. Постолит, А.В. Visual Studio .NET: разработка приложений баз данных / А.В. Постолит. — Спб. : Бхв-Петербург, 2003. — 544 с.

5. Практическое использование ADO.NET. Доступ к данным в Internet / Ш. Вилдермьюс. — М. : Издательский дом «Вильямс», 2003. — 288 с.

6. Язык Java. Курс программирования / Савитч Уолтер. — 2-е изд. — М. : Издательский дом «Вильямс», 2002. — 928 с.

УЧЕБНОЕ ИЗДАНИЕ

Евгений Евгеньевич Бизянов
Николай Николаевич Кононенко

БАЗЫ ДАННЫХ

Лабораторный практикум

В авторской редакции

Художественное оформление обложки Н. В. Чернышова

Заказ № 310. Формат 60x84 $^{1}/_{16}$.
Бумага офс. Печать RISO.
Усл. печат. л. 10,8 Уч.-изд. л. 9,3
Издательство не несет ответственность за содержание
материала, предоставленного автором к печати.
Издатель и изготовитель:
ГОУ ВПО ЛНР «Донбасский государственный технический университет»
пр. Ленина, 16, г. Алчевск, ЛНР, 94204
(ИЗДАТЕЛЬСКО-ПОЛИГРАФИЧЕСКИЙ ЦЕНТР, ауд. 2113, т/факс 2-58-59)
Свидетельство о государственной регистрации издателя, изготовителя
и распространителя средства массовой информации
МИ-СГР ИД 000055 от 05.02.2016