

Документ подписан простой электронной подписью

Информация о владельце:

ФИО: Федорова Марина Владимировна

Должность: Директор филиала

Дата подписания: 29.09.2020 16:08:58

Уникальный программный ключ:

e766def0e2eb4560a114a7e458117041ca

МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА  
ФИЛИАЛ ФЕДЕРАЛЬНОГО СУДАРСТВЕННОГО БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО  
УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ  
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ  
СООБЩЕНИЯ» в г. Алатыре

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ  
ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ  
ПО ПРОФЕССИОНАЛЬНОМУ МОДУЛЮ  
ПМ.11 РАЗРАБОТКА, АДМИНИСТРИРОВАНИЕ  
И ЗАЩИТА БАЗ ДАННЫХ  
ДЛЯ СТУДЕНТОВ СПЕЦИАЛЬНОСТИ  
09.02.07 ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ПРОГРАММИРОВАНИЕ**

*Базовая подготовка  
среднего профессионального образования*

Алатырь 2020



## СОДЕРЖАНИЕ

## ВВЕДЕНИЕ

Выполнение лабораторных работ должно способствовать закреплению и углублению знаний, полученных в процессе изучения лекционных курсов по дисциплинам «Базы данных», «Системы управления базами данных». Рассмотрен весь процесс проектирования реляционной базы данных от построения инфологической модели до ее конкретной реализации с использованием популярной СУБД MySQL.

Практикум знакомит студента с созданием баз данных и таблиц, их заполнением, извлечением и удалением записей. Рассмотрены встроенные функции, транзакции, временные таблицы, хранимые процедуры, триггеры, курсоры. Описаны способы обеспечения целостности и безопасности данных

Лабораторный практикум включает 10 лабораторных работ, в каждой из которых кратко представлен теоретический материал по рассматриваемой теме. Рассмотрены конкретные примеры, приведены варианты заданий для самостоятельного выполнения. При описании интерфейса использована версия СУБД MySQL 5.0.

Полученные знания, умения и навыки могут быть использованы при создании баз данных для экономических информационных систем, разрабатываемых студентами в процессе дипломного проектирования.

По результатам лабораторного практикума оформляется отчет в виде пояснительной записки объемом 20–25 страниц формата А4. Текст готовится с помощью текстового процессора Microsoft Word. Размеры полей страницы: левое – 25 мм, правое – 10 мм, верхнее – 20 мм, нижнее – 25 мм. Размер шрифта – 14 пунктов. Тип шрифта – Times New Roman. Величина абзацного отступа – 10 мм. Межстрочный интервал – одинарный. Каждый раздел записки начинается с новой страницы, номера страниц располагаются внизу страницы по центру. Рисунки и таблицы нумеруются последовательно арабскими цифрами.

## 1. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

*Базу данных (БД)* можно определить как унифицированную совокупность данных, совместно используемую различными приложениями в рамках некоторой единой автоматизированной информационной системы.

Программное обеспечение, осуществляющее операции над БД, получило название СУБД – *система управления базами данных*.

*Запрос* – специальным образом описанное требование, определяющее состав производимых над БД операций по выборке или модификации хранимых данных.

Для подготовки запросов чаще всего используют *структурированный язык запросов – SQL (Structured Query Language)*. Этот язык стал фактическим стандартом языка работы с реляционными БД. Он является непроцедурным языком и не содержит операторов управления, организации подпрограмм, ввода-вывода и т. д. Поэтому SQL автономно не используется, а обычно погружен в среду встроенного языка программирования СУБД или процедурного языка (типа C++ или Pascal).

Современные СУБД позволяют создавать запросы, не применяя SQL. Однако его применение позволяет расширить возможности СУБД.

Категории SQL-запросов:

*определение данных (Data Definition Language, DDL)* – SQL-запросы, позволяющие пользователям создавать и модифицировать структуру объектов БД (таблицы, представления и индексы); команды DDL влияют на контейнеры, содержащие данные, а не на данные;

*запросы данных (Data Query Language, DQL)* – включает выражения SQL для получения данных из базы;

*манипуляции с данными (Data Manipulation Language, DML)* – SQL-запросы, позволяющие пользователю добавлять и удалять данные (в форме строк), а также модифицировать имеющиеся в БД;

*контроль данных (Data Control Language, DCL)* – SQL-запросы, позволяющие администраторам контролировать доступ к данным в базе и использовать различные системные привилегии СУБД;

*контроль транзакций* – набор команд, которые пользователь применяет для того, чтобы вся транзакция либо была успешно выполнена, либо нет; команды контроля транзакций не вполне соответствуют синтаксису SQL-запросов, но положительно влияют на выполнение запросов, включенных в транзакцию.

Архитектура современных профессиональных СУБД базируется на принципах *клиент-серверного взаимодействия* программных компонентов. Сервер – процесс, обслуживающий информационную потребность клиента.

Клиент – приложение, посылающее запрос на обслуживание сервером. Клиент инициирует связь с сервером, определяет вид запроса, получает от сервера результат обслуживания, подтверждает окончание обслуживания. Поскольку стандартом интерфейса «клиент-сервер» в этом случае является язык SQL, СУБД называют *SQL-сервером*.

На клиентском компьютере может выполняться *SQL-клиент* – программа, предоставленная поставщиком СУБД и обеспечивающая пользователю возможность вводить SQL-запросы, посылать их в СУБД и просматривать результат.

По пользовательскому интерфейсу SQL-клиенты разделяются на три типа:

*клиент с интерфейсом командной строки* – команды вводятся с клавиатуры как текст, клиент можно использовать в любой операционной системе;

*клиент с графическим интерфейсом пользователя (GUI, Graphical User Interface)* – выполняется в оконной системе (Microsoft Windows) и отображает данные, используя графические элементы (значки, кнопки и диалоговые окна);

*клиент с Web-интерфейсом* – выполняется на сервере БД, а для взаимодействия с пользователем используется Web-браузер на клиентском компьютере.

Одним из наиболее популярных SQL-серверов БД является *MySQL* – небольшая и надежная реляционная СУБД с возможностью отката и восстановления после сбоя, многопользовательская, многопоточная, с высокой производительностью. Сервер MySQL предназначен как для критических по задачам производственных систем с большой нагрузкой, так и для встраивания в программное обеспечение массового распространения.

MySQL – открытое программное обеспечение (распространяется с открытым исходным кодом). Благодаря высокой производительности и простоте настройки, богатому выбору API-интерфейсов, а также функциональным средствам работы с сетями, сервер MySQL стал одним из самых удачных вариантов для разработки Web-приложений, взаимодействующих с БД.

Система MySQL может быть реализована как:

автономная настольная система;

клиент-серверная система.

Если MySQL используется как автономная настольная система, то клиентское приложение исполняется на том же компьютере, на котором хранится программное обеспечение MySQL и БД. Сетевые соединения от клиента к серверу не устанавливаются. Настольные системы полезны в следующих случаях:

при доступе к БД лишь одного пользователя;

при небольшом числе пользователей, работающих с БД не одновременно.

*Клиент-серверная система* может иметь:

двухзвенную установку;

трехзвенную установку.

Независимо от варианта установки, программное обеспечение и базы данных MySQL размещаются на центральном компьютере (*сервере баз данных*). Пользователи работают на компьютерах-клиентах. Доступ пользователей к серверу БД производится при помощи:

приложений с компьютеров-клиентов (в двухзвенных системах);

приложений, выполняющихся на специальном компьютере – сервере приложений (в трехзвенных системах).

В двухзвенных системах клиенты исполняют приложения, осуществляющие доступ к серверу БД непосредственно через сеть. Клиенты называются толстыми,

поскольку выполняют два вида работы:

- исполняют программный код, соответствующий функциональным задачам;
- исполняют код, отображающий результаты доступа к БД.

Двухзвенная установка полезна при небольшом количестве пользователей, потому что для соединения с каждым из пользователей расходуются системные ресурсы (память и блокировки). Чем больше количество соединений с пользователями, тем хуже производительность системы из-за соперничества за ресурсы.

В трехзвенных системах в задачи компьютеров-клиентов входит лишь исполнение программного кода по вызову функций сервера приложений и отображение результатов. Такие клиенты называются тонкими. Сервер приложений исполняет многопоточные приложения, с которыми могут работать много пользователей одновременно. Сервер приложений соединяется с сервером БД, осуществляет доступ к данным и возвращает результаты клиенту.

С распространением Интернета клиенты и серверы стали взаимодействовать в глобальной сети. Web-среда предоставила пользователям дружелюбный интерфейс, за формирование которого отвечает Web-сервер. Такой подход позволил использовать для работы с удаленными БД Web-браузер, не прибегая к услугам специфических клиентских программ. Например, клиенты торговой компании, желающие ознакомиться со списком товаров, используют браузер для посещения сайта компании. Web-страницу со списком товаров формирует специальный модуль (скрипт), выполняющийся на Web-сервере компании. Для получения информации этот скрипт посылает SQL-запросы СУБД, находящейся на сервере БД.

Таким образом, в трехуровневой архитектуре Интернета выделяются:

- клиент – Web-браузер (клиентское приложение), который взаимодействует с Web-сервером, посылая ему запросы на отображение той или иной Web-страницы;

- Web-сервер – на котором выполняется Web-приложение, формирующее SQL-запрос к СУБД (которая должна вернуть необходимые данные из БД);

- сервер баз данных – на котором размещены СУБД и база данных.

## 2. УСТАНОВКА MYSQL

С Web -страницы <http://dev.mysql.com/downloads/> можно загрузить дистрибутив MySQL. Для загрузки доступны:

- MySQL 5.1 – рекомендуемая версия (релиз);

- MySQL 5.4 – версия, находящаяся в стадии бета-тестирования;

- MySQL 6 – версия, находящаяся в стадии альфа-тестирования;

- MySQL 4.1 – устаревшая, но поддерживаемая версия.

Когда разрабатываемая версия переходит в стадию релиза, в нее прекращают добавлять нововведения и лишь исправляют найденные ошибки. Все нововведения добавляются в новую версию. Поддержка старой рекомендуемой версии прекращается. Справочное руководство можно найти по адресу <http://dev.mysql.com/doc/>.

На открывшейся странице будет представлен список дистрибутивов, скомпилированных под разные операционные системы. Для Windows предлагаются:

*Windows Essentials (x86)* – урезанная версия дистрибутива, из которой удалены все вспомогательные утилиты («голый» сервер MySQL);

*Windows (x86)* – полная версия, включающая автоматический установщик;

*Without installer (unzip in C:\)* – полная версия дистрибутива без автоматического установщика.

Рекомендуется выбрать дистрибутив Windows (x86). Можно загрузить графические клиенты для работы с MySQL-сервером (MySQL Administrator, MySQL Query Browser, MySQL Migration Toolkit), которые свободно распространяются на сайте <http://dev.mysql.com/downloads/gui-tools/5.0.html>.

При работе в Windows NT/2000/XP/Server 2003 необходимо войти в систему с привилегиями администратора, разархивировать дистрибутив *mysql-5.0.51b-win32.zip* во временный каталог, после чего запустить файл *setup.exe*. Для продолжения установки следует нажать кнопку *Next*, после чего откроется окно, в котором предлагается тип инсталляции:

*Typical* – устанавливаются сервер MySQL, клиент командной строки *mysql* и утилиты командной строки;

*Complete* – устанавливаются все компоненты (эталонный набор, встроенный сервер библиотеки, поддержка скриптов, документация);

*Custom* – предоставляет возможность выбора необходимых пакетов и изменения инсталляционного пути.

После завершения настройки выводится завершающее окно. Если вас удовлетворяют параметры настройки, то нажмите кнопку *Install*.

По окончании инсталляции можно зарегистрироваться на Web-сайте MySQL. Регистрация дает возможность участвовать в форумах – *forums.mysql.com*, сообщать об ошибках – *bugs.mysql.com* и подписаться на информационный бюллетень. Для получения подробной информации, нажмите *More*, для продолжения – *Next*. Заключительный экран инсталлятора сообщает об окончании установки.

Для запуска Мастера Конфигурации нужно поставить галочку в пункт *Configure the MySQL Server now*. К настройке всегда можно вернуться, выбрав пункт системного меню *Пуск > Программы > MySQL > MySQL Server 5.0 > MySQL Server Instance Config Wizard*. Рекомендуется сразу произвести настройку.

Настройка начинается со стартового окна. После нажатия кнопки *Next* открывается окно, в котором предлагается выбрать тип конфигурации.

Доступны два типа конфигурации:

*Detailed Configuration* (детализированная конфигурация) – предназначена для опытных пользователей, которые хотят сконфигурировать сервер, учитывая возможности компьютера и конкретные задачи;

*Standard Configuration* (стандартная конфигурация) – предназначена для новых пользователей, которым нужно быстро установить MySQL, не вникая в детали конфигурации сервера.

Для гибкой настройки системы следует выбрать пункт *Detailed Configuration*.

После нажатия кнопки *Next* открывается окно настройки производительности MySQL. В этом окне есть три опции:

*Developer Machine* (машина разработчика) – типичная настольная рабочая станция, на которой MySQL предназначен только для личного использования и на которой выполняется множество других приложений; сервер MySQL будет сконфигурирован для использования минимальных системных ресурсов;

*Server Machine* (сервер) – машина, на которой сервер MySQL выполняется вместе с другими приложениями-серверами (FTP, e-mail, Web-серверы); сервер MySQL будет сконфигурирован для использования умеренной части ресурсов;

*Dedicated MySQL Server Machine* (выделенный сервер) – машина выполняет только функции выделенного сервера MySQL и никакие другие приложения на ней не выполняются; сервер MySQL будет сконфигурирован для использования всех доступных системных ресурсов.

Опции различаются по интенсивности использования процессора, объема оперативной памяти и жесткого диска. Следует выбрать первый пункт.

Следующее окно позволяет выбрать предпочтительный тип для таблиц, который назначается по умолчанию. В этом окне есть три опции:

*Multifunctional Database* (многофункциональная БД) – допускается использование двух механизмов памяти – *InnoDB* и *MyISAM*, при этом ресурсы равномерно разделяются между ними; рекомендуется для пользователей, использующих оба механизма памяти на регулярной основе;

*Transactional Database Only* (только транзакционная БД) – допускается использование обоих механизмов памяти – *InnoDB* и *MyISAM*, но большинство ресурсов выделяется механизму *InnoDB*; рекомендуется для пользователей, почти исключительно использующих *InnoDB* и минимально использующих *MyISAM*;

*Non-Transactional Database Only* (только нетранзакционная БД) – отключается механизм памяти *InnoDB* и все ресурсы выделяются механизму памяти *MyISAM*; рекомендуется для пользователей, не использующих *InnoDB*.

Следует выбрать первый пункт. Результатом работы утилиты *MySQL Server Instance Config Wizard* является конфигурационный файл *my.ini*, который всегда можно отредактировать вручную (можно скорректировать тип таблиц).

Можно определить местонахождение файлов таблиц *InnoDB*, если в системе есть более надежное устройство хранения данных (система RAID). Выбор диска и пути к файлам осуществляется в следующем окне.

Следующее окно предлагает выбрать максимальное число клиентов, которые могут одновременно подключиться к серверу. Первый пункт (рекомендуется) предполагает число соединений не больше 20, второй пункт устанавливает предел на 500 соединений, а третий пункт позволяет назначить предел.

Следующее окно позволяет разрешить или отключить организацию сети *TCP/IP* и конфигурировать порт, используемый для соединения с сервером (по умолчанию – 3306). Здесь же можно включить и отключить строгий режим, который заставляет MySQL быть похожим на другие СУБД. Для приложений, рассчитанных на «прощающее» поведение MySQL, этот режим можно отключить.

В следующем окне устанавливается кодировка по умолчанию:  
*стандартная кодировка символов* – по умолчанию используется набор символов *latin1* (для английского и многих западноевропейских языков);  
*улучшенная многоязычная поддержка* – набор символов Unicode, который может использовать символы из множества различных языков;  
*ручной выбор кодировки символов* – установка кодировки символов вручную.

Необходимо отметить третий пункт и в выпадающем списке выбрать пункт *cp1251*, соответствующий русской Windows-кодировке.

В среде Windows можно установить MySQL в качестве службы, что обеспечит его запуск при старте системы и корректное завершение работы при выключении компьютера. Сервер MySQL может быть запущен автоматически при старте системы и перезапущен автоматически в случае отказа службы. Следующее окно предназначено для настройки службы. Флажок *Install As Windows Service* позволяет установить службу с именем, которое можно выбрать в выпадающем списке.

Чтобы не запускать сервер MySQL автоматически, удалите галочку из опции *Автоматический запуск сервера MySQL*. Флажок *Include Bin Directory in Windows PATH* позволяет прописать путь к каталогу MySQL в системной переменной *PATH*, что удобно при частом использовании утилит из этого каталога.

В следующем окне производится настройка учетных записей. Если вы не знакомы с системой авторизации MySQL и производите установку первый раз, рекомендуется снять флажок *Modify Security Settings*.

После нажатия кнопки *Execute* на конечной странице утилиты настройки MySQL-сервера *MySQL Server Instance Config Wizard* будет создан конфигурационный файл *my.ini* и запущен сервер MySQL.

После установки MySQL в меню *Пуск* будет создан новый раздел *MySQL* со следующими пунктами:

- MySQL Command Line Client* – клиент командной строки MySQL;
- MySQL Server Instance Config Wizard* – мастер конфигурации сервера;
- MySQL Manual* – руководство пользователя (ссылка на документацию).

После установки и конфигурирования MySQL необходимо убедиться в его работоспособности. Пройдите путь *Пуск > Все программы > MySQL > MySQL Server 5.0 > MySQL Command Line Client*. В окне DOS введите пароль, который установлен при инсталляции, и нажмите *Enter* (если пароль не установлен – сразу *Enter*).

Если при запуске клиента в командной строке выводится сообщение «*Character set 'cp1251' is not a compiled character set and is not specified in the 'C:\mysql\share\charsets\Index.xml' file*», звучит сигнал и окно закрывается, необходимо исправить конфигурационный файл *my.ini*. Директива *default-character-set=cp1251* должна присутствовать только в секции *[mysqld]* и отсутствовать в секции *[mysql]* (там ее следует закомментировать – поставить впереди символ # ).

### 3. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

#### Лабораторная работа № 1

#### Проектирование базы данных с использованием ER-технологии

##### Теоретические сведения

которыми должен следовать пробел. Кроме того, MySQL содержит ряд собственных комментариев. Shell-комментарий # действует аналогично – все, что расположено правее его, является текстом комментария. C-комментарий /\* \*/ является многострочным – комментарий начинается с /\* и заканчивается, когда встретится завершение \*/.

Создание и выбор базы данных. Осуществляется с помощью оператора

`create database имя_базы_данных;`

Убедиться в том, что оператор выполнил задачу, можно с помощью оператора `show databases;`

Теедметной области;

анализ связей и определение их характеристик – степени связи, мощности и класса принадлежности;

построение набора предварительных отношений с указанием предполагаемого первичного ключа для каждого отношения;

подготовка списка всех неключевых атрибутов и назначение каждого из этих атрибутов одному из предварительных отношений;

проверка нахождения всех полученных отношений в нормальной форме Бойса-Кодда;

построение модели данных.

##### Практическая работа

При выполнении лабораторной работы необходимо:

для своего варианта, соответствующего определенной предметной области, построить логическую модель данных в соответствии со стандартом IDEF1X;

построить физическую модель;

составить отчет по лабораторной работе.

##### Пример выполнения работы

Особенности диалекта SQL в СУБД MySQL рассмотрим на примере учебной базы данных *book* Интернет-магазина, торгующего компьютерной литературой. В базе данных должна поддерживаться следующая информация:

тематические каталоги, по которым сгруппированы книги;

предлагаемые книги (название, автор, год издания, цена, имеющееся на складе количество);

зарегистрированные покупатели (имя, отчество, фамилия, телефон, адрес электронной почты, статус – авторизованный, неавторизованный, заблокированный, активный с хорошей кредитной историей);

покупки, совершенные в магазине (время совершения покупки, число приобретенных экземпляров книги).

Логическая модель данных предметной области в стандарте IDEF1X представлена на рис. 1. Выделены сущности *КАТАЛОГ*, *КНИГА*, *КЛИЕНТ*, *ЗАКАЗ*, между которыми установлены

неидентифицирующие связи мощностью один-ко-многим, определенные спецификой предметной области.



Рис. 1. Логическая модель данных предметной области

Физическая модель данных предметной области в стандарте IDEF1X для целевой СУБД MySQL представлена на рис. 2.

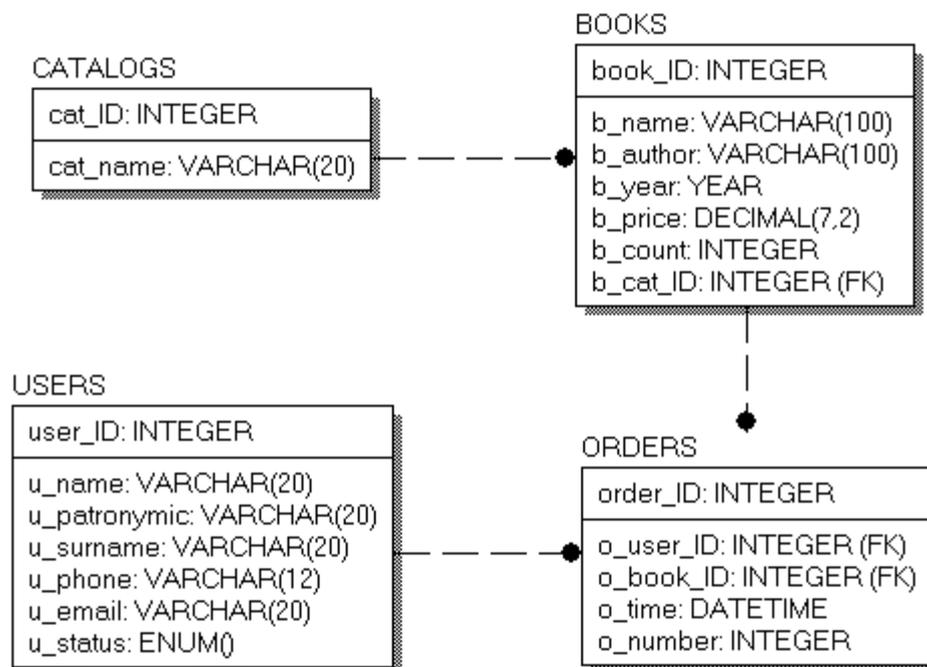


Рис. 2. Физическая модель предметной области

База данных *book* состоит из четырех таблиц:  
*catalogs* – список торговых каталогов;  
*books* – список предлагаемых книг;

*users* – список зарегистрированных пользователей магазина;

*orders* – список заказов (осуществленных сделок).

Таблица *catalogs* состоит из двух полей:

*cat\_ID* – уникальный код каталога;

*cat\_name* – имя каталога.

Оба поля должны быть снабжены атрибутом *NOT NULL*, поскольку неопределенное значение для них недопустимо.

Таблица *books* состоит из семи полей:

*book\_ID* – уникальный код книги;

*b\_name* – название книги;

*b\_author* – автор книги;

*b\_year* – год издания;

*b\_price* – цена книги;

*b\_count* – количество книг на складе;

*b\_cat\_ID* – код каталога из таблицы *catalogs*.

Цена книги *b\_price* и количество экземпляров на складе *b\_count* могут иметь атрибут *NULL*. На момент доставки часто неизвестны количество товара и его цена, но отразить факт наличия товара в прайс-листе необходимо.

Поле *b\_cat\_ID* устанавливает связь между таблицами *catalogs* и *books*. Это поле должно быть объявлено как внешний ключ (FK) с правилом каскадного удаления и обновления. Обновление таблицы *catalogs* вызовет автоматическое обновление таблицы *books*. Удаление каталога в таблице *catalogs* приведет к автоматическому удалению всех записей в таблице *books*, соответствующих каталогу.

Таблица *users* состоит из семи полей:

*user\_ID* – уникальный код покупателя;

*u\_name* – имя покупателя;

*u\_patronymic* – отчество покупателя;

*u\_surname* – фамилия покупателя;

*u\_phone* – телефон покупателя (если имеется);

*u\_email* – e-mail покупателя (если имеется);

*u\_status* – статус покупателя.

Статус покупателя представлен полем типа *ENUM*, которое может принимать одно из четырех значений:

*active* – авторизованный покупатель, который может осуществлять покупки через Интернет;

*passive* – неавторизованный покупатель (значение по умолчанию), который осуществил процедуру регистрации, но не подтвердил ее и пока не может осуществлять покупки через Интернет, однако ему доступны каталоги для просмотра;

*lock* – заблокированный покупатель, не может осуществлять покупки и просматривать каталоги магазина;

*gold* – активный покупатель с хорошей кредитной историей, которому предоставляется скидка при следующих покупках в магазине.

Поля *u\_phone* и *u\_email* могут быть снабжены атрибутом *NULL*. Остальные поля должны получить атрибут *NOT NULL*.

Таблица *orders* включает пять полей:

*order\_ID* – уникальный номер сделки;

*o\_user\_ID* – номер пользователя из таблицы *users*;

*o\_book\_ID* – номер товарной позиции из таблицы *books*;

*o\_time* – время совершения сделки;

*o\_number* – число приобретенных товаров.

Поля таблицы *orders* должны быть снабжены атрибутом *NOT NULL*, т. к. при совершении покупки вся информация должна быть занесена в таблицу.

В таблице *orders* устанавливается связь с таблицами *users* (за счет поля *o\_user\_ID*) и *books* (за счет поля *o\_book\_ID*). Эти поля объявлены как внешние ключи (FK) с правилом каскадного удаления и обновления. Обновление таблиц *users* и *books* приведет к автоматическому обновлению таблицы *orders*. Удаление любого пользователя в таблице *users* приведет к автоматическому удалению всех записей в таблице *orders*, соответствующих этому пользователю.

## Лабораторная работа № 2

### Создание и связывание таблиц базы данных в среде MySQL

#### Теоретические сведения

Рассмотрим следующие вопросы:

создание и выбор базы данных;

создание таблиц;

столбцы и типы данных в MySQL;

создание индексов;

удаление таблиц, индексов и баз данных;

изменение структуры таблиц.

Базы данных, таблицы и индексы легко создаются в рамках графического интерфейса MySQL, но мы будем использовать монитор MySQL (клиент командной строки), чтобы лучше понять структуру БД, таблиц и индексов.

#### Чувствительность к регистру и идентификаторы.

Имена БД подчиняются тем же правилам зависимости от регистра символов, каким следуют каталоги операционной системы. Имена таблиц следуют тем же правилам, что и имена файлов. Все остальное не зависит от регистра.

Все идентификаторы, кроме имен псевдонимов, могут содержать до 64 символов. Имена псевдонимов могут иметь до 255 символов.

Идентификаторы могут содержать любые допустимые символы, но имена баз данных не могут содержать символы /, \ и ., а имена таблиц – символы . и /.

Зарезервированные слова можно использовать для идентификаторов, если заключить их в кавычки.

**Комментарий в SQL.** Начинается с двух дефисов (--), за которыми должен следовать пробел. Кроме того, MySQL содержит ряд собственных комментариев.

Shell-комментарий # действует аналогично – все, что расположено правее его, является текстом комментария. С-комментарий /\* \*/ является многострочным – комментарий начинается с /\* и заканчивается, когда встретится завершение \*/.

**Создание и выбор базы данных.** Осуществляется с помощью оператора *CREATE DATABASE имя\_базы\_данных;*

Убедиться в том, что оператор выполнил задачу, можно с помощью оператора *SHOW DATABASES;*

Теперь имеется пустая БД, ожидающая создания таблиц. Прежде чем работать с БД, необходимо выбрать эту БД с помощью оператора *USE имя\_базы\_данных;*

Теперь все действия по умолчанию будут применяться именно к этой БД.

**Создание таблиц.** Используется оператор *CREATE TABLE*, который в общем виде выглядит следующим образом:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]  
        имя_таблицы (определение таблицы)  
[type=тип_таблицы];
```

Ключевое слово *TEMPORARY* используется для создания таблиц, которые будут существовать только в текущем сеансе работы с БД и будут автоматически удалены, когда сеанс завершится.

При использовании выражения *IF NOT EXISTS* таблица будет создана только в том случае, если еще нет таблицы с указанным именем.

```
create [temporary] table [if not exists]  
имя_таблицы LIKE имя_старой_таблицы;
```

После имени таблицы в скобках объявляются имена столбцов, их типы и другая информация. В определении столбца можно добавить следующие описания.

Объявить для любого столбца *NOT NULL* или *NULL* (столбцу запрещено или не запрещено содержать значения *NULL*). По умолчанию – *NULL*.

Объявить для столбца значение по умолчанию, используя ключевое слово *DEFAULT*, за которым должно следовать значение по умолчанию.

Использовать ключевое слово *AUTO\_INCREMENT*, чтобы генерировать порядковый номер. Автоматически генерируемое значение будет на единицу большим, чем наибольшее значение в таблице. Первая введенная строка будет иметь порядковый номер 1. В таблице можно иметь не более одного столбца *AUTO\_INCREMENT*, и он должен индексироваться.

Объявить столбец первичным ключом таблицы с помощью выражения *PRIMARY KEY*.

Объявить столбец внешним ключом, используя выражение *FOREIGN KEY*, с ссылкой на соответствующую таблицу с помощью выражения *REFERENCES*.

Индексировать столбец с помощью слов *INDEX* или *KEY* (синонимы). Такие столбцы не обязательно должны содержать уникальные значения.

Индексировать столбец с помощью слова *UNIQUE*, которое используется для указания того, что столбец должен содержать уникальные значения.

Создать полнотекстовые индексы на основе столбцов типа *TEXT*, *CHAR* или *VARCHAR* с помощью слова *FULLTEXT* (только с таблицами *MyISAM*).

После закрывающей скобки можно указать тип таблицы:

MyISAM – таблицы этого типа являются «родными» для MySQL, работают очень быстро и поддерживают полнотекстовую индексацию;

InnoDB – ACID-совместимый механизм хранения, поддерживающий транзакции, внешние ключи, каскадное удаление и блокировки на уровне строк;

BDB (Berkeley DB) – является механизмом хранения, который обеспечивает поддержку транзакций и блокировки на уровне страниц;

MEMORY (HEAP) – таблицы целиком хранятся в оперативной памяти и никогда не записываются на диск, поэтому работают очень быстро, но ограничены в размерах и не допускают возможности восстановления в случае отказа системы;

MERGE – тип позволяет объединить несколько таблиц MyISAM с одной структурой, чтобы с ним можно было направлять запросы как к одной таблице;

NDB Cluster – тип предназначен для организации кластеров MySQL, когда таблицы распределены между несколькими компьютерами, объединенными в сеть;

ARCHIVE – тип введен для хранения большого объема данных в сжатом формате; таблицы поддерживают только два SQL-оператора: INSERT и SELECT, причем оператор SELECT выполняется по методу полного сканирования таблицы;

CSV – формат представляет собой обычный текстовый файл, записи в котором хранятся в строках, а поля разделены точкой с запятой (широко распространен в компьютерном мире, любая программа, поддерживающая CSV-формат, может открыть такой файл);

FEDERATED – тип позволяет хранить данные в таблицах на другой машине сети (при создании таблицы в локальной директории создается только файл определения структуры таблицы, а все данные хранятся на удаленной машине).

MySQL поддерживает следующие типы данных, допустимые для столбцов:

числовые;

строковые;

календарные;

null – специальный тип, обозначающий отсутствие информации.

Числовые типы используются для хранения чисел и представляют два подтипа:

точные числовые типы;

приближенные числовые типы.

К точным числовым типам (табл. 1) относятся целый тип INTEGER и его вариации, а также вещественный тип decimal (синонимы numeric и dec). Последний используется для представления денежных данных.

Числовые типы могут характеризоваться максимальной длиной M. Для типа decimal параметр m задает число символов для отображения всего числа, а d – для его дробной части. Например: b\_price DECIMAL (5, 2). Цифра 5 определяет общее число символов под число, а цифра 2 – количество знаков после запятой (интервал величин от -99.99 до 99.99). Можно не использовать параметры вообще, указать только общую длину или указать длину и число десятичных разрядов.

Объявления точных числовых типов можно завершать ключевыми словами UNSIGNED и (или) ZEROFILL. Ключевое слово UNSIGNED указывает, что столбец содержит только положительные числа или нули. Ключевое слово ZEROFILL означает, что число будет отображаться с ведущими нулями.

Таблица 1

Тип	Объем памяти	Диапазон
<i>TINYINT (M)</i> <i>TINYINT UNSIGNED</i>	1 байт	от -128 до 127 (от $-2^7$ до $2^7-1$ ) от 0 до 255 (от 0 до $2^8-1$ )
<i>SMALLINT (M)</i> <i>SMALLINT UNSIGNED</i>	2 байта	от -32 768 до 32 767 (от $-2^{15}$ до $2^{15}-1$ ) от 0 до 65 535 (от 0 до $2^{16}-1$ )
<i>MEDIUMINT (M)</i> <i>MEDIUMINT UNSIGNED</i>	3 байта	от -8 388 608 до 8 388 607 (от $-2^{23}$ до $2^{23}-1$ ) от 0 до 16 777 215 (от 0 до $2^{24}-1$ )
<i>INT (INTEGER) (M)</i> <i>INT UNSIGNED</i>	4 байта	от -2 147 683 648 до 2 147 683 647 (от $-2^{31}$ до $2^{31}-1$ ) от 0 до 4 294 967 295 (от 0 до $2^{32}-1$ )
<i>BIGINT (M)</i> <i>BIGINT UNSIGNED</i>	8 байт	(от $-2^{63}$ до $2^{63}-1$ ) (от 0 до $2^{64}-1$ )
<i>BIT (M)</i>	$(M+7)/8$ байт	От 1 до 64 битов, в зависимости от значения <i>M</i>
<i>BOOL, BOOLEAN</i>	1 байт	0 ( <i>false</i> ) либо 1 ( <i>true</i> )
<i>DECIMAL (M, D),</i> <i>NUMERIC (M, D)</i>	<i>M + 2</i> байта	Повышенная точность, зависит от параметров <i>M</i> и <i>D</i>

К приближенным числовым типам (табл. 2) относятся:

*FLOAT* – представление чисел с плавающей запятой с обычной точностью;

*DOUBLE* – представление чисел с плавающей запятой с двойной точностью.

Таблица 2

Тип	Объем памяти	Диапазон
<i>FLOAT (M, D)</i>	4 байта	Минимальное по модулю значение $1.175494351 \cdot 10^{-39}$ Максимальное по модулю значение $3.402823466 \cdot 10^{38}$
<i>DOUBLE (M, D),</i> <i>REAL (M,D),</i> <i>DOUBLE PRECISION (M,D)</i>	8 байт	Минимальное по модулю значение $2.2250738585072014 \cdot 10^{-308}$ Максимальное по модулю значение $1.797693134862315 \cdot 10^{308}$

Числовые типы с плавающей точкой также могут иметь параметр *UNSIGNED*. Атрибут предотвращает хранение в столбце отрицательных величин, но максимальный интервал величин столбца остается прежним.

Приближенные числовые данные могут задаваться в обычной форме (например, 45.67) и в форме с плавающей точкой (например, 5.456E-02 или 4.674E+04).

Текстовые типы и строки (табл. 3):

- *CHAR* – хранение строк фиксированной длины;
- *VARCHAR* – хранение строк переменной длины;
- *TEXT, BLOB* и их вариации – хранение больших фрагментов текста;
- *ENUM* и *SET* – хранение значений из заданного списка.

• Таблица 3

Тип	Объем памяти	Максимальный размер
-----	--------------	---------------------

<i>CHAR(M)</i>	<i>M</i> символов	<i>M</i> символов
232-1 символов <i>VARCHAR(M)</i>	<i>L+1</i> символов	<i>M</i> символов
<i>ENUM('value 1', 'value2 ', ...)</i> <i>L+4</i> символов <i>TINYBLOB, TINYTEXT</i> <i>L+1</i> символов <i>28-1</i> символов  <i>BLOB, TEXT</i> <i>L+2</i> символов <i>216-1</i> символов  <i>MEDIUMBLOB,</i> <i>MEDIUMTEXT</i> <i>L+3</i> символов <i>224-1</i> символов  <i>LOB, LONGTEXT</i>	1 или 2 байта	65 535 элементов
<i>SET('value 1', 'value2', ...)</i>	1, 2, 3, 4 или 8 байт	64 элемента

- Здесь *l* – длина хранимой в ячейке строки, а приплюсованные к *l* байты – накладные расходы для хранения длины строки.

- Для строк *varchar* требуется количество символов, равное длине строки плюс 1 байт, тогда как тип *char(m)*, независимо от длины строки, использует для ее хранения все *m* символов. Тип *char* обрабатывается эффективнее переменных типов. Нельзя смешивать в таблице столбцы *char* и *varchar*. Если есть столбец переменной длины, все столбцы типа *char* будут приведены к типу *varchar*.

- Типы *blob* и *text* аналогичны и отличаются в деталях. При выполнении операций над столбцами типа *text* учитывается кодировка, а типа *blob* – нет. Тип *text* используется для хранения больших объемов текста, тип *blob* – для больших двоичных объектов (электронные документы, изображения, звук). Основное отличие *text* от *char* и *varchar* – поддержка полнотекстового поиска.

- Строки типов данных *enum* и *set* принимают значения из заданного списка. Значение типа *enum* должно содержать точно одно значение из указанного множества, тогда как столбцы *set* могут содержать любой или все элементы заданного множества одновременно. Для типа *set* (как и для *enum*) при объявлении задается список возможных значений, но ячейка может принимать любое значение из списка, а пустая строка означает, что ни один из элементов списка не выбран.

- Типы *enum* и *set* задаются списком строк, но во внутреннем представлении элементы множеств сохраняются в виде чисел. Элементы типа *enum* нумеруются последовательно, начиная с 1. Под столбец может отводиться 1 байт (до 256 элементов в списке) или 2 байта (от 257 до 65536 элементов в списке). Элементы типа *set* обрабатываются как биты, размер типа определяется числом элементов в списке: 1 байт (от 1 до 8 элементов), 2 байта (от 9 до 16 элементов), 3 байта (от 17 до 24 элементов), 4 байта (от 25 до 32 элементов) и 8 байт (от 33 до 64 элементов).

- Календарные типы данных (табл. 4):

- Date – для хранения даты (формат YYYY-MM-DD для дат вида 2009-10-15 и формат YY-MM-DD для дат вида 09-10-15);
- Time – для хранения времени суток (формат hh:mm:ss, где hh – часы, mm – минуты, ss – секунды, например, 10:48:56);
- Datetime– для представления и даты, и времени суток;
- Timestamp – если в соответствующем столбце строки не указать конкретное значение или NULL, там будет записано время, когда соответствующая строка была создана или в последний раз изменена (в формате DATETIME);
- Year – позволяет хранить только год.

• Таблица 4

Тип	Объем памяти	Диапазон
DATE	3 байта	от '1000-01-01' до '9999-12-31'
TIME	3 байта	от '-828:59:59' до '828:59:59'
DATETIME	8 байт	от '1000-01-01 00:00:00' до '9999-12-31 00:00:00'
TIMESTAMP (M)	4 байта	от '1970-01-01 00:00:00' до '2038-12-31 59:59:59'
YEAR(2) YEAR(4)	1 байт	формат YY, диапазон – от 1970 до 2069 формат YYYY, диапазон – от 1901 до 2155

- Дни, месяцы, часы, минуты и секунды можно записывать как с ведущим нулем, так и без него. Например, все следующие записи идентичны:
  - '2009-04-06 02:04:08'      '2009-4-06 02:04:08'      '2009-4-6 02:04:08'
  - '2009-4-6 2:04:08'      '2009-4-6 2:4:08'      '2009-4-6 2:4:8'
- В качестве разделителя между годами, месяцами, днями, часами, минутами, секундами может выступать любой символ, отличный от цифры. Так, следующие значения идентичны:
  - '09-12-31 11:30:45' '09.12.31 11+30+45' '09/12/31 11\*30\*45'
- При указании времени после секунд через точку можно указать микросекунды, т. е. использовать расширенный формат вида hh:mm:ss.ffffff, например '10:25:14.000001'. Кроме того, можно использовать краткие форматы HH:MM и HH (вместо пропущенных величин будут подставлены нулевые значения).
- Если время задается в недопустимом формате, то в поле записывается нулевое значение. Нулевое значение присваивается полям временного типа по умолчанию, когда им не присваивается иницилирующее значение (табл. 5).

• Таблица 5

Тип	Нулевое значение
DATE	'0000-00-00'
TIME	'00:00:00'
DATETIME	'0000-00-00 00:00:00'
TIMESTAMP	0000000000000000
YEAR	0000

- Формат *TIMESTAMP* совпадает с *DATETIME*, но во внутреннем представлении дата хранится как число секунд, прошедших с полуночи 1 января 1970 г. (такое исчисление принято в операционной системе UNIX, а дата 01.01.1970 считается началом эпохи UNIX и днем рождения операционной системы).

- Если в таблице несколько столбцов *TIMESTAMP*, при модификации записи текущее время будет записываться только в один из них (первый). Можно явно указать столбец, которому необходимо назначать текущую дату при создании или изменении записи. Чтобы поля принимали текущую дату при создании записи, следует после определения столбца добавить *DEFAULT CURRENT\_TIMESTAMP*. Если текущее время должно выставляться при модификации записи, при использовании оператора *UPDATE* следует добавить *ON UPDATE CURRENT\_TIMESTAMP*.

- Тип данных *NULL* используется, когда информации недостаточно и для части данных нельзя определить, какое значение они примут. Для указания того, что поле может принимать неопределенное значение, в определении столбца после типа данных следует указать ключевое слово *NULL*. Если поле не должно принимать значение *NULL*, следует указать ключевое слово *NOT NULL*.

- Рекомендации по выбору типа данных.

- Обработка числовых данных происходит быстрее строчковых. Так как типы *ENUM* и *SET* имеют внутреннее числовое представление, им следует отдавать предпочтение перед другими видами строчковых данных, если это возможно.

- Производительность можно увеличить за счет представления строк в виде чисел. Пример – преобразование IP-адреса из строки в *BIGINT*. Это позволит уменьшая *u\_name* из таблицы *users*:

- `create index name on users (u_name);`

- Перед ключевым словом *index* может присутствовать *UNIQUE*, требующее уникальности ограничения.

- Корректность таблиц в БД можно проверить с помощью оператора

- `SHOW TABLES;`

- Более подробную информацию о структуре таблицы дает команда

- `DESCRIBE имя_таблицы;`

- Переименование БД. Специального оператора переименования БД нет, но можно переименовать каталог БД в системном каталоге (...\*DATA*).

- Удаление БД. Удалить всю БД вместе с ее содержимым можно командой с данными переменной длины неизбежно, для дефрагментации таблицы следует применять команду *OPTIMIZE TABLE*.

- **Обеспечение ссылочной целостности.** Задается конструкцией:

- `FOREIGN KEY [name_key] (col1, ... ) REFERENCES tbl (tbl_col, ... )`

- *[ON DELETE {CASCADE | SET NULL | NO ACTION | RESTRICT | SET DEFAULT}] [ON UPDATE {CASCADE | SET NULL | NO ACTION | RESTRICT | SET DEFAULT}]*

- Конструкция позволяет задать внешний ключ с необязательным именем *name\_key* на столбцах, которые задаются в круглых скобках (один или несколько). Ключевое слово *REFERENCES* указывает таблицу *tbl*, на которую ссылается внешний ключ, в круглых скобках указываются имена столбцов. Необязательные конструкции *ON DELETE* и *ON UPDATE* позволяют задать поведение СУБД при удалении и обновлении строк из таблицы-предка. Параметры, следующие за этими ключевыми словами, имеют следующие значения:

- *CASCADE* – при удалении или обновлении записи в таблице-предке, содержащей первичный ключ, записи со ссылками на это значение в таблице-потомке удаляются или обновляются автоматически;

- *SET NULL* – при удалении или обновлении записи в таблице-предке, содержащей первичный ключ, в таблице-потомке значения внешнего ключа, ссылающегося на таблицу-предка, устанавливаются в *NULL*;

- *NO ACTION* – при удалении или обновлении записей, содержащих первичный ключ, с таблицей-потомком никаких действий не производится;

- *RESTRICT* – если в таблице-потомке имеются записи, ссылающиеся на первичный ключ таблицы-предка, при удалении или обновлении записей с таким первичным ключом возвращается ошибка;

- *SET DEFAULT* – согласно стандарту *SQL*, при удалении или обновлении первичного ключа в таблице-потомке для ссылающихся на него записей в поле внешнего ключа должно устанавливаться значение по умолчанию (в *MySQL* это ключевое слово зарезервировано, но не обрабатывается).

- **Создание индексов.** Индексы играют большую роль в БД, т. к. это основной способ ускорения их работы. Записи в таблице располагаются хаотически. Чтобы найти нужную запись, необходимо сканировать всю таблицу, на что уходит много времени. Идея индексов состоит в том, чтобы создать для столбца копию, которая постоянно будет поддерживаться в отсортированном состоянии. Это позволяет быстро осуществлять поиск по такому столбцу.

- Все необходимые индексы формируются при создании таблицы. Индексированы будут все столбцы, объявленные как *PRIMARY KEY*, *KEY*, *UNIQUE* или *INDEX*. Индекс также можно добавить с помощью оператора *CREATE INDEX*. Перед выполнением оператор преобразуется в оператор *ALTER TABLE*. Например, создание индекса с именем *name* на основе поля *u\_name* из таблицы *users*:

- *CREATE INDEX name ON users (u\_name);*

- Перед ключевым словом *INDEX* может присутствовать *UNIQUE*, требующее уникальности ограничения.

- Корректность таблиц в БД можно проверить с помощью оператора

- *SHOW TABLES*;
- Более подробную информацию о структуре таблицы дает команда
  - *DESCRIBE имя\_таблицы*;
  - **Переименование БД.** Специального оператора переименования БД нет, но можно переименовать каталог БД в системном каталоге (... \DATA).
    - **Удаление БД.** Удалить всю БД вместе с ее содержимым можно командой:
      - *DROP DATABASE [IF EXISTS] имя\_базы\_данных*;
      - **Удаление таблиц и индексов.** Удалить таблицу можно с помощью оператора:
        - *DROP TABLE [IF EXISTS] имя\_таблицы*;
        - Удалить индекс можно с помощью оператора:
          - *DROP INDEX имя\_индекса ON имя\_таблицы*;
        - **Изменение структуры таблиц.** Изменить структуру существующей таблицы можно с помощью оператора *ALTER TABLE*. Например, можно создать индекс *name* для таблицы *users* следующим образом:
          - *ALTER TABLE users ADD INDEX name (u\_name)*;
          - Оператор *ALTER TABLE* является исключительно гибким, поэтому он имеет огромное множество дополнительных ключевых слов.
- **Практическая работа**
- При выполнении лабораторной работы необходимо для заданной предметной области средствами MySQL:
  - создать базу данных;
  - создать таблицы, определить поля таблиц, индексы;
  - определить связи между таблицами и ограничения целостности;
  - составить отчет по лабораторной работе.
- **Пример выполнения работы**
  - Операторы создания БД *book* имеют следующий вид (целесообразно создать в *Блокноте* текстовый файл и записать туда эти операторы *INSERT INTO books VALUES (1,'JavaScript в кармане','Рева О.Н.', 2008, 42.00, 10, 1)*;
  - *INSERT INTO books VALUES (2,'Visual FoxPro 9.0','Клепинин В.Б.', 2007, 660.00, 2, 1)*;
  - *INSERT INTO books VALUES (3,'C++ Как он есть','Тимофеев В.В.',2009, 218.00, 4, 1)*;
  - *INSE22R22T22 22I22N22T22O22 22b22o22o22k22s22 22V22A22L22U22E22S22*

23(23423,23'23!>740=85                    23?@8;>65=89                    23A                    23?><>ILN  
23C23#23'23,23'23\$0@>=>2                    23.23.23'23,23                    23223023023823,23  
23123623923.23023023,23 23123,23 23123)23;23

• 23I23N23S23E23R23T23                    23I23N23T23O23                    23b23o23o23k23s23  
23V23A23L23U23E23S23                    23(23523,23'23D23e23l23p23h23i23.23                    230@>4=K5  
23A>25BK'23,23'23(:@K;L

23.23.23'23,23223023023723,23223423323.23023023,23623,23123)23;23

• 23I23N23S23E23R23T23                    23I23N23T23O23                    23b23o23o23k23s23  
23V23A23L23U23E23S23                    23(23623,23'23D23e23l23p23h23i23.23                    23>;=>5  
23@C:>2>4AB2>'23,23'23!CE0@52 23.23'23,23223023023r(12) NULL,

- u\_email varchar(20) NULL,
- u\_status ENUM ('active','passive','lock','gold') default 'passive',
- PRIMARY KEY (user\_ID)
- ) TYPE=InnoDB;

• CREATE TABLE orders (  
• order\_ID int(6) NOT NULL AUTO\_INCREMENT,  
• o\_user\_ID int NOT NULL,  
• o\_book\_ID int NOT NULL,  
• o\_time datetime NOT NULL default '0000-00-00 00:00:00',  
• o\_number int(6) NOT NULL default '0',  
• PRIMARY KEY (order\_ID),  
• FOREIGN KEY (o\_book\_ID) REFERENCES books(book\_ID) ON DELETE  
CASCADE ON UPDATE CASCADE,  
• FOREIGN KEY (o\_user\_ID) REFERENCES users(user\_ID) ON DELETE  
CASCADE ON UPDATE CASCADE  
• )TYPE=InnoDB;

• **Лабораторная работа № 3**  
**Вставка, удаление и обновление данных**

- **Теоретические сведения**
- Рассмотрим следующие вопросы:
  - вставка данных с помощью оператора *INSERT*;
  - удаление данных операторами *DELETE* и *TRUNCATE*;
  - обновление данных с помощью оператора *UPDATE*.

• После создания БД и таблиц перед разработчиком встает задача заполнения таблиц данными. В реляционных БД традиционно применяют три подхода:

- однострочный оператор *INSERT* – добавляет в таблицу новую запись;
- многострочный оператор *INSERT* – добавляет в таблицу несколько записей;
- пакетная загрузка *LOAD DATA INFILE* – добавление данных из файла.

• **Вставка данных с помощью оператора *INSERT*.**  
Однострочный оператор *INSERT* может использоваться в нескольких

формах. Упрощенный синтаксис первой формы:

- *INSERT [IGNORE] [INTO] имя\_таблицы [(имя\_столбца, ... )]*
- *VALUES (выражение, ... );*

• Оператор вставляет новую запись в таблицу *имя\_таблицы*. Значения полей записи перечисляются в списке (*выражение, ...* ). Порядок следования столбцов задается списком (*имя\_столбца, ...* ). Список столбцов (*имя\_столбца, ...* ) позволяет менять порядок следования столбцов при добавлении.

• Первичный ключ таблицы является уникальным, и попытка добавить уже существующее значение приведет к ошибке. Чтобы новые записи с дублирующим ключом отбрасывались без генерации ошибки, следует добавить после оператора *INSERT* ключевое слово *IGNORE*.

• Другая форма оператора *INSERT* предполагает использование слова *SET*:

- *INSERT [IGNORE] [INTO] имя\_таблицы*
- *SET имя\_столбца1 = выражение1, имя\_столбца2 = выражение2, ... ;*

• Оператор заносит в таблицу *имя\_таблицы* новую запись, столбец *имя\_столбца* в которой получает значение *выражение*.

• Многострочный оператор *INSERT* совпадает по форме с однострочным оператором, но после ключевого слова *VALUES* добавляется через запятую несколько списков (*выражение, ...* ).

• Практические примеры использования оператора *INSERT* для заполнения учебной БД *book* см. ниже, в пункте «Пример выполнения работы».

• **Удаление данных.** Для удаления записей из таблиц предусмотрены:

- оператор *DELETE*;
- оператор *TRUNCATE TABLE*.

- Оператор *DELETE* имеет следующий синтаксис:
- *DELETE FROM имя\_таблицы*
- *[WHERE условие]*
- *[ORDER BY имя\_поля]*
- *[LIMIT число\_строк];*

• Оператор удаляет из таблицы *имя\_таблицы* записи, удовлетворяющие условию. В следующем примере из таблицы *catalogs* удаляются записи, имеющие значение первичного ключа *catalog\_id* больше двух.

- Если в операторе отсутствует условие *WHERE*, удаляются все записи таблицы.

```
mysql> DELETE FROM catalogs;
Query OK, 2 rows affected (0.03 sec)
```

```
mysql> SELECT * FROM catalogs;
Empty set (0.00 sec)
```

- Ограничение *LIMIT* позволяет задать максимальное число записей, которые могут быть удалены. Следующий запрос удаляет все записи таблицы *orders*, но не более 3 записей.

```
mysql> DELETE FROM orders LIMIT 3;
Query OK, 3 rows affected (0.01 sec)
```

```
mysql> SELECT * FROM orders;
```

order_ID	o_user_ID	o_book_ID	o_time	o_number
4	4	20	2009-03-10 18:20:00	1
5	3	20	2009-03-17 19:15:36	1

```
2 rows in set (0.00 sec)
```

- Конструкция *ORDER BY* обычно применяется вместе с ключевым словом *LIMIT*. Например, если необходимо удалить 20 первых записей таблицы, то производится сортировка по полю типа *DATETIME* – тогда в первую очередь будут удалены самые старые записи.

- Оператор *TRUNCATE TABLE* полностью очищает таблицу и не допускает условного удаления. Он аналогичен оператору *DELETE* без условия *WHERE* и ограничения *LIMIT*. Удаление происходит гораздо быстрее, т. к. осуществляется не перебор записей, а полное очищение таблицы.

```
mysql> TRUNCATE TABLE orders;
Query OK, 5 rows affected (0.03 sec)
```

```
mysql> SELECT * FROM orders;
Empty set (0.00 sec)
```

- **Обновление данных.** Обновление данных (изменение значений полей в существующих записях) обеспечивают:

- оператор *UPDATE*;
- оператор *REPLACE*.

- Оператор *UPDATE* позволяет обновлять отдельные поля в существующих записях. Имеет следующий синтаксис
  - *UPDATE [IGNORE] имя\_таблицы*
  - *SET имя\_столбца1= выражение1 [, имя\_столбца2 = выражение2 ...]*
  - *[WHERE условие]*
  - *[ORDER BY имя\_поля]*
  - *[LIMIT число\_строк]* ;
- После ключевого слова *UPDATE* указывается таблица, которая изменяется. В предложении *SET* указывается, какие столбцы обновляются и устанавливаются их новые значения. Необязательное условие *WHERE* позволяет задать критерий отбора строк (обновляться будут только строки, удовлетворяющие условию).
- Если указывается необязательное ключевое слово *IGNORE*, то команда обновления не будет прервана, даже если при обновлении возникнет ошибка дублирования ключей. Строки, породившие конфликтные ситуации, обновлены не будут.
- Запрос, изменяющий в таблице *catalogs* «Сети» на «Компьютерные сети».

```
mysql> UPDATE catalogs SET cat_name='Компьютерные сети'
      -> WHERE cat_name='Сети';
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> SELECT * FROM catalogs;
+----+-----+
| cat_ID | cat_name          |
+----+-----+
| 1      | Программирование |
| 2      | Интернет         |
| 3      | Базы данных      |
| 4      | Компьютерные сети |
| 5      | Мультимедиа     |
+----+-----+
```

• 5 rows in set (0.00 sec)

- Обновлять можно всю таблицу. Пусть требуется уменьшить на 5 % цену на все книги. Для этого следует старую цену в рублях умножить на 0,95.

```
mysql> UPDATE books SET b_price=b_price*0.95;
Query OK, 30 rows affected (0.03 sec)
Rows matched: 30 Changed: 30 Warnings: 0
```

```
mysql> SELECT book_ID, b_name, b_price FROM books;
```

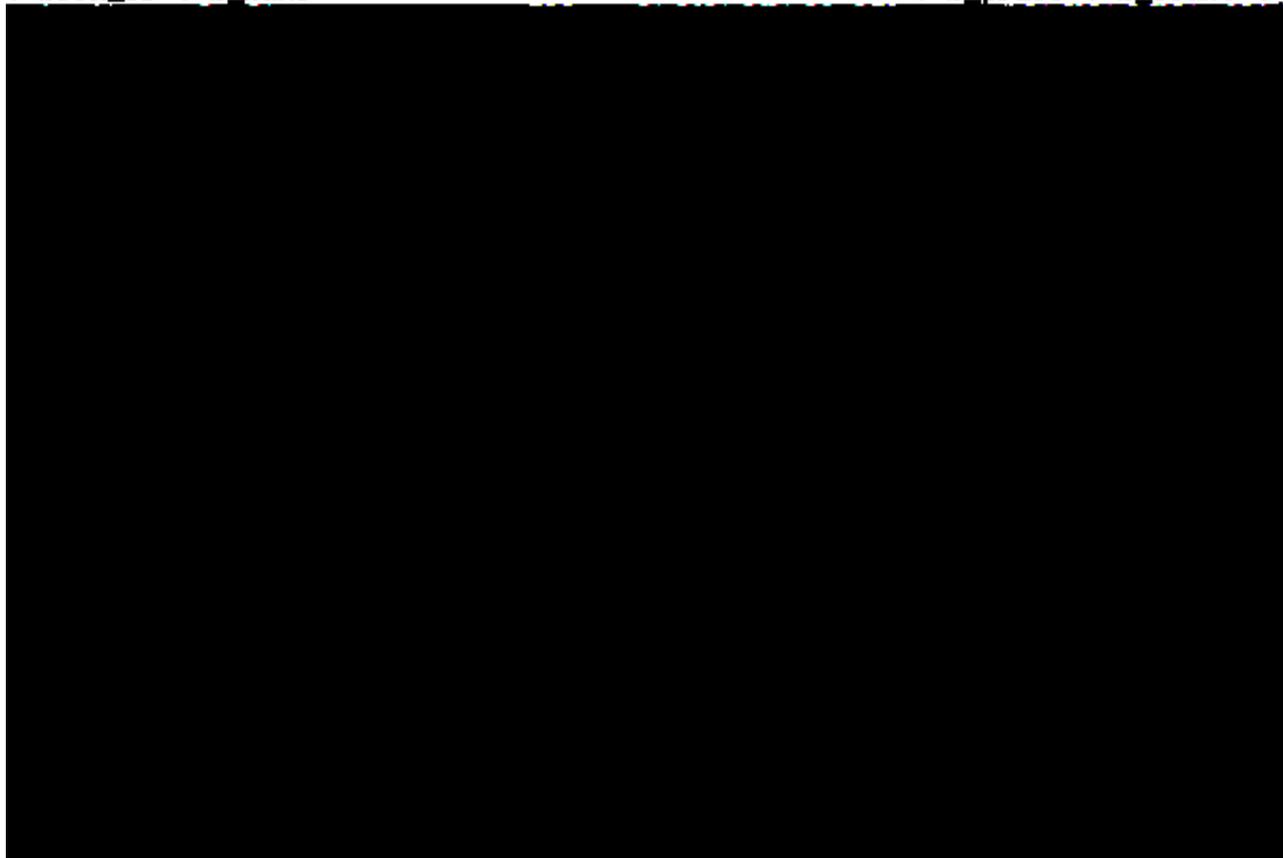
book_ID	b_name	b_price
1	JavaScript в кармане	39.90
2	Visual FoxPro 9.0	627.00
3	C++ Как он есть	207.10
4	Создание приложений с помощью С#	160.55
5	Delphi. Народные советы	230.85
6	Delphi. Полное руководство	475.00
7	Профессиональное программирование на PHP	293.55
8	Совершенный код	732.45
9	Практика программирования	203.30
10	Принципы маршрутизации в Internet	406.60
11	Поиск в Internet	101.65
12	Web-конструирование	168.15
13	Самоучитель Интернет	114.95
14	Популярные интернет-браузеры	77.90
15	Общение в Интернете	80.75
16	Базы данных	309.70
17	Базы данных. Разработка приложений	179.55
18	Раскрытие тайн SQL	190.00
19	Практикум по Access	82.65
20	Компьютерные сети	598.50
21	Сети. Поиск неисправностей	412.30
22	Безопасность сетей	438.90

- Инструкции *LIMIT* и *ORDER BY* позволяют ограничить число изменяемых записей. При этом за один запрос можно обновить несколько столбцов таблицы. Например, необходимо в таблице *books* для десяти самых дешевых товарных позиций уменьшить количество книг на складе на единицу, а цену – на 5 %.

```
mysql> UPDATE books SET b_price=b_price*0.95,b_count=b_count-1
-> ORDER BY b_price LIMIT 10;
Query OK, 10 rows affected (0.05 sec)
Rows matched: 10 Changed: 10 Warnings: 0
```

```
mysql> SELECT book_ID, b_name, b_price, b_count FROM books;
```

```
+-----+-----+-----+-----+
| book_ID | b_name | b_price | b_count |
```



- Оператор *REPLACE* работает как оператор *INSERT*, за исключением того, что старая запись с тем же значением индекса *UNIQUE* или *PRIMARY KEY* перед внесением новой будет удалена. Если не используются индексы *UNIQUE* или *PRIMARY KEY*, то применение оператора *REPLACE* не имеет смысла.

- Синтаксис оператора *REPLACE* аналогичен синтаксису оператора *INSERT*:

- *REPLACE [INTO] имя\_таблицы [(имя\_столбца, ... )]*
- *VALUES (выражение, ... )*

- В таблицу вставляются значения, определяемые в списке после ключевого слова *VALUES*. Задать порядок столбцов можно при помощи необязательного списка, следующего за именем таблицы. Как и оператор *INSERT*, оператор *REPLACE* допускает многострочный формат.

•

### • Практическая работа

- При выполнении лабораторной работы необходимо для заданной предметной области средствами MySQL:

- заполнить согласованными данными таблицы БД;

- при необходимости исправить введенную информацию;
- составить отчет по лабораторной работе.

- **Пример выполнения работы**

- Операторы заполнения БД *book* имеют следующий вид.
- *USE book;*
- *SET CHARACTER SET cp1251;*
- *DELETE FROM catalogs;*
- *INSERT INTO catalogs VALUES (1,'Программирование');*
- *INSERT INTO catalogs VALUES (2,'Интернет');*
- *INSERT INTO catalogs VALUES (3,'Базы данных');*
- *INSERT INTO catalogs VALUES (4,'Сети');*
- *INSERT INTO catalogs VALUES (5,'Мультимедиа');*
- *DELETE FROM books;*
- *INSERT INTO books VALUES (1,'JavaScript в кармане','Рева О.Н.', 2008, 42.00, 10, 1);*
  - *INSERT INTO books VALUES (2,'Visual FoxPro 9.0','Клепинин В.Б.', 2007, 660.00, 2, 1);*
  - *INSERT INTO books VALUES (3,'C++ Как он есть','Тимофеев В.В.',2009, 218.00, 4, 1);*
  - *INSERT INTO books VALUES (4,'Создание приложений с помощью C#','Фаронов В.В.', 2008, 169.00, 1, 1);*
  - *INSERT INTO books VALUES (5,'Delphi. Народные советы','Шкрыль А.А.',2007,243.00,6,1);*
  - *INSERT INTO books VALUES (6,'Delphi. Полное руководство','Сухарев М.',2008,500.00,6,1);*
  - *INSERT INTO books VALUES (7,'Профессиональное программирование на PHP', 'Шлосснейгл Дж.', 2006, 309.00, 5, 1);*
  - *INSERT INTO books VALUES (8,'Совершенный код','Макконнелл С.', 2007, 771.00, 1, 1);*
  - *INSERT INTO books VALUES (9,'Практика программирования','Керниган Б.', 2004, 214.00, 12, 1);*
  - *INSERT INTO books VALUES (10,'Принципы маршрутизации в Internet','Хелеби С.', 2001, 428.00, 4, 2);*
  - *INSERT INTO books VALUES (11,'Поиск в Internet','Гусев В.С.',2004,107.00,2,2);*
  - *INSERT INTO books VALUES (12,'Web-конструирование','Дуванов А.А.', 2003, 177.00, 6, 2);*
  - *INSERT INTO books VALUES (13,'Самоучитель Интернет','Константинов Ю.П.', 2009, 121.00, 4, 2);*
  - *INSERT INTO books VALUES (14,'Популярные интернет-браузеры','Маринин С.А.', 2007, 82.00, 6, 2);*
  - *INSERT INTO books VALUES (15,'Общение в Интернете','Экслер А.', 2006, 85.00, 5, 2);*
  - *INSERT INTO books VALUES (16,'Базы данных','Малыхина*

М.П.', 2006, 326.00, 2, 3);

- `INSERT INTO books VALUES (17,'Базы данных. Разработка приложений', 'Рудикова Л.В.', 2006, 189.00, 6, 3);`

- `INSERT INTO books VALUES (18,'Раскрытие тайн SQL', 'Онпель Э.', 2007, 200.00, 3, 3);`

- `INSERT INTO books VALUES (19,'Практикум по Access', 'Золотова С.И.', 2007, 87.00, 6, 3);`

- `INSERT INTO books VALUES (20,'Компьютерные сети', 'Танненбаум Э.', 2007, 630.00, 6, 4);`

- `INSERT INTO books VALUES (21,'Сети. Поиск неисправностей', 'Бигелоу С.', 2005, 434.00, 4, 4);`

- `INSERT INTO books VALUES (22,'Безопасность сетей', 'Брегг Р.', 2006, 462.00, 5, 4);`

- `INSERT INTO books VALUES (23,'Анализ и диагностика компьютерных сетей', 'Хогдал Дж.', 2001, 344.00, 3, 4);`

- `INSERT INTO books VALUES (24,'Локальные вычислительные сети', 'Епанешников А.', 2005, 82.00, 8, 4);`

- `INSERT INTO books VALUES (25,'Цифровая фотография', 'Надеждин Н.', 2004, 149.00, 20, 5);`

- `INSERT INTO books VALUES (26,'Музыкальный компьютер для гитариста', 'Петелин Р.Ю.', 2004, 217.00, 15, 5);`

- `INSERT INTO books VALUES (27,'Видео на ПК', 'Федорова А.', 2003, 231.00, 10, 5);`

- `INSERT INTO books VALUES (28,'Мультипликация во Flash', 'Киркпатрик Г.', 2006, 211.00, 20, 5);`

- `INSERT INTO books VALUES (29,'Запись CD и DVD', 'Гультияев А.К.', 2003, 167.00, 12, 5);`

- `INSERT INTO books VALUES (30,'Запись и обработка звука на компьютере', 'Лоянич А.А.', 2008, 51.00, 8, 5);`

- `DELETE FROM users;`

- `INSERT INTO users VALUES (1,'Александр', 'Валерьевич', 'Иванов', '58-98-78', 'ivanov@email.ru', 'active');`

- `INSERT INTO users VALUES (2,'Сергей', 'Иванович', 'Лосев', '90-57-77', 'losev@email.ru', 'passive');`

- `INSERT INTO users VALUES (3,'Игорь', 'Николаевич', 'Симонов', '95-66-61', 'simonov@email.ru', 'active');`

- `INSERT INTO users VALUES (4,'Максим', 'Петрович', 'Кузнецов', NULL, 'kuznetsov@email.ru', 'active');`

- `INSERT INTO users VALUES (5,'Анатолий', 'Юрьевич', 'Петров', NULL, NULL, 'lock');`

- `INSERT INTO users VALUES (6,'Александр', 'Александрович', 'Корнеев', '89-78-36', 'korneev@email.ru', 'gold');`

- *DELETE FROM orders;*
- *INSERT INTO orders VALUES (1,3,8,'2009-01-04 10:39:38',1);*
- *INSERT INTO orders VALUES (2,6,10,'2009-02-10 09:40:29',2);*
- *INSERT INTO orders VALUES (3,1,20,'2009-02-18 13:41:05',4);*
- *INSERT INTO orders VALUES (4,4,20,'2009-03-10 18:20:00',1);*
- *INSERT INTO orders VALUES (5,3,20,'2009-03-17 19:15:36',1);*

- **Лабораторная работа № 4**

- **Создание простых запросов на выборку**

- - **Теоретические сведения**
  - Рассмотрим следующие вопросы:
    - выборка данных из одной таблицы с помощью оператора *SELECT*;
    - использование в запросах операторов и встроенных функций MySQL.

- Для выполнения запросов (извлечения строк из одной или нескольких таблиц БД) используется оператор *SELECT*. Результатом запроса всегда является таблица. Результаты запроса могут быть использованы для создания новой таблицы. Таблица, полученная в результате запроса, может стать предметом дальнейших запросов.

- Общая форма оператора *SELECT*:
- *SELECT столбцы FROM таблицы*
- *[WHERE условия]*
- *[GROUP BY группа [HAVING групповые\_условия] ]*
- *[ORDER BY имя\_поля]*
- *[LIMIT пределы];*

- Оператор *SELECT* имеет много опций. Их можно использовать или не использовать, но они должны указываться в том порядке, в каком они приведены. Если требуется вывести все столбцы таблицы, необязательно перечислять их после ключевого слова *SELECT*, достаточно заменить этот список символом *\**.

```
mysql> SELECT * FROM orders;
```

order_ID	o_user_ID	o_book_ID	o_time	o_number
1	3	8	2009-01-04 10:39:38	1
2	6	10	2009-02-10 09:40:29	2
3	1	20	2009-02-18 13:41:05	4
4	4	20	2009-03-10 18:20:00	1
5	3	20	2009-03-17 19:15:36	1

```
5 rows in set (0.02 sec)
```

- Список столбцов в операторе *SELECT* используют, если нужно изменить порядок следования столбцов в результирующей таблице или выбрать часть столбцов.

```
mysql> SELECT cat_name, cat_ID FROM catalogs;
```

cat_name	cat_ID
Программирование	1
Интернет	2
Базы данных	3
Сети	4
Мультимедиа	5

```
5 rows in set (0.01 sec)
```

- **Условия выборки.** Гораздо чаще встречается ситуация, когда необходимо изменить количество выводимых строк. Для выбора записей, удовлетворяющих определенным критериям поиска, можно использовать конструкцию *WHERE*.

- В запросе можно использовать ключевое слово *DISTINCT*, чтобы результат не содержал повторений уже имеющихся значений, например:

```
mysql> SELECT DISTINCT u_status FROM users;
```

u_status
active
passive
lock
gold

- **Сортировка.** Результат выборки – записи, расположенные в том порядке, в котором они хранятся в БД. Чтобы отсортировать значения по одному из столбцов, необходимо после конструкции *ORDER BY* указать этот столбец, например:

- Сортировку записей можно производить по нескольким столбцам (их следует указать после слов *ORDER BY* через запятую). Число столбцов, указываемых в конструкции *ORDER BY*, не ограничено.

- По умолчанию сортировка производится в прямом порядке (записи располагаются от наименьшего значения поля сортировки до наибольшего). Обратный порядок сортировки реализуется с помощью ключевого слова *DESC*:

```
mysql> SELECT o_time FROM orders ORDER BY o_time DESC;
```

o_time
2009-03-17 19:15:36
2009-03-10 18:20:00
2009-02-18 13:41:05
2009-02-10 09:40:29
2009-01-04 10:39:38

```
5 rows in set (0.27 sec)
```

- Для прямой сортировки существует ключевое слово *ASC*, но так как записи сортируются в прямом порядке по умолчанию, данное ключевое слово опускают.

- **Ограничение выборки.** Результат выборки может содержать тысячи записей, вывод и обработка которых занимают значительное время. Поэтому информацию часто разбивают на страницы и предоставляют ее пользователю частями. Постраничная навигация используется при помощи ключевого слова *LIMIT*, за которым следует число выводимых записей. Следующий запрос извлекает первые 5 записей, при этом осуществляется обратная сортировка по полю *b\_count*:

```
mysql> SELECT book_ID, b_count FROM books
-> ORDER BY b_count DESC
-> LIMIT 5;
```

book_ID	b_count
28	20
25	20
26	15
29	12
9	12

• 5 rows in set (0.03 sec)

- Для извлечения следующих пяти записей используется ключевое слово *LIMIT* с двумя цифрами. Первая указывает позицию, начиная с которой необходимо вернуть результат, вторая цифра – число извлекаемых записей, например:

```
mysql> SELECT book_ID, b_count FROM books
-> ORDER BY b_count DESC
-> LIMIT 5,5;
```

book_ID	b_count
1	10
27	10
24	8
30	8
20	6

• 5 rows in set (0.00 sec)

- При определении смещения нумерация строк начинается с нуля (поэтому в последнем примере для шестой строки указано смещение 5).

- **Группировка записей.** Конструкция *GROUP BY* позволяет группировать извлекаемые строки. Она полезна в комбинации с функциями, применяемыми к группам строк. Эти функции (табл. 6) называются агрегатами (суммирующими функциями) и вычисляют одно значение для каждой группы, создаваемой конструкцией *GROUP BY*. Функции позволяют узнать число строк в группе, подсчитать среднее значение, получить сумму значений столбцов. Результирующее значение рассчитывается для значений, не равных *NULL* (исключение – функция *COUNT(\*)*). Допустимо использование

этих функций в запросах без группировки (вся выборка – одна группа).

- Пример использования функции `COUNT( )`, которая возвращает число строк в таблице, значения указанного столбца для которых отличны от `NULL`:

```
mysql> SELECT COUNT(book_ID) FROM books;
+-----+
| COUNT(book_ID) |
+-----+
|                30 |
+-----+
1 row in set (0.16 sec)
```

• Таблица 6

Обозначение	Описание
<code>AVG ( [DISTINCT] expr )</code>	Возвращает среднее значение аргумента <i>expr</i> . В качестве аргумента обычно выступает имя столбца. Необязательное слово <code>DISTINCT</code> позволяет обрабатывать только уникальные значения столбца <i>expr</i>
<code>COUNT ( )</code>	Подсчитывает число записей и имеет несколько форм. Форма <code>COUNT (выражение)</code> возвращает число записей в таблице, поле <i>выражение</i> для которых не равно <code>NULL</code> . Форма <code>COUNT(*)</code> возвращает общее число строк в таблице независимо от того, принимает какое-либо поле значение <code>NULL</code> или нет. Форма <code>COUNT (DISTINCT выражение1, выражение2, ... )</code> позволяет использовать ключевое слово <code>DISTINCT</code> , которое позволяет подсчитать только уникальные значения столбца
<code>MIN ( [DISTINCT] expr )</code>	Возвращает минимальное значение среди всех непустых значений выбранных строк в столбце <i>expr</i> . Необязательное слово <code>DISTINCT</code> позволяет обрабатывать только уникальные значения столбца <i>expr</i>
<code>MAX ( [DISTINCT] expr )</code>	Возвращает максимальное значение среди всех непустых значений выбранных строк в столбце <i>expr</i> . Необязательное слово <code>DISTINCT</code> позволяет обрабатывать только уникальные значения столбца <i>expr</i>
<code>STD (expr)</code>	Возвращает стандартное среднеквадратичное отклонение в аргументе <i>expr</i>
<code>STDDEV_SAMP (expr)</code>	Возвращает выборочное среднеквадратичное отклонение в аргументе <i>expr</i>
<code>SUM ( [DISTINCT] expr )</code>	Возвращает сумму величин в столбце <i>expr</i> . Необязательное слово <code>DISTINCT</code> позволяет обрабатывать только уникальные значения столбца <i>expr</i>

- Использование ключевого слова `DISTINCT` с функцией `COUNT( )` позволяет вернуть число уникальных значений `b_cat_ID` в таблице `books`, например:

```
mysql> SELECT COUNT(DISTINCT b_cat_ID) FROM books;
+-----+
| COUNT(DISTINCT b_cat_ID) |
+-----+
|                5 |
+-----+
1 row in set (0.02 sec)
```

- В *SELECT*-запросе столбцу можно назначить новое имя с помощью оператора *AS*. Например, результату функции *COUNT()* присваивается псевдоним *total*:

```
mysql> SELECT COUNT(order_ID) AS total FROM orders;
```

```
+-----+
| total |
+-----+
|      5 |
+-----+
```

• 1 row in set (0.05 sec)

- Использование функций в конструкции *WHERE* приведет к ошибке. В следующем примере показана попытка извлечения из таблицы *catalogs* записи с максимальным значением поля *cat\_ID*:

```
mysql> SELECT * FROM catalogs WHERE cat_ID=MAX(cat_ID);
```

• ERROR 1111 (HY000): Invalid use of group function

- Решение задачи следует искать в использовании конструкции *ORDER BY*:

```
mysql> SELECT * FROM catalogs ORDER BY cat_ID DESC LIMIT 1;
```

```
+-----+-----+
| cat_ID | cat_name |
+-----+-----+
|      5 | Музыка |
+-----+-----+
```

• 1 row in set (0.00 sec)

- Для извлечения уникальных записей используют конструкцию *GROUP BY* с именем столбца, по которому группируется результат:

```
mysql> SELECT b_cat_ID FROM books
-> GROUP BY b_cat_ID ORDER BY b_cat_ID;
```

```
+-----+
| b_cat_ID |
+-----+
|         1 |
|         2 |
|         3 |
|         4 |
|         5 |
+-----+
```

• 5 rows in set (0.03 sec)

- При использовании *GROUP BY* возможно использование условия *WHERE*:

```
mysql> SELECT b_cat_ID, COUNT(b_cat_ID) FROM books
-> WHERE b_cat_ID > 2
-> GROUP BY b_cat_ID
-> ORDER BY b_cat_ID;
```

```
+-----+-----+
| b_cat_ID | COUNT(b_cat_ID) |
+-----+-----+
|         3 |                4 |
|         4 |                5 |
|         5 |                6 |
+-----+-----+
```

• 3 rows in set (0.02 sec)

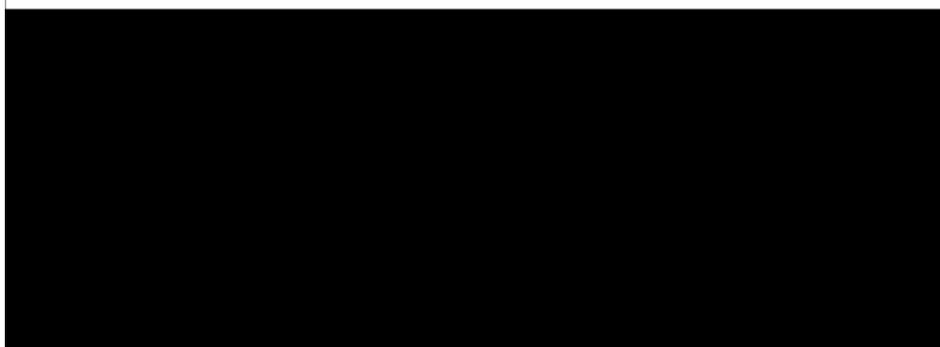
- Часто при задании условий требуется ограничить выборку по результату функции (например, выбрать каталоги, где число товарных позиций больше 5). Использование для этих целей конструкции *WHERE* приводит к ошибке. Для решения этой проблемы вместо ключевого слова *WHERE* используется ключевое слово *HAVING*, располагающееся за конструкцией *GROUP BY*:

- ```
mysql> SELECT b_cat_ID, COUNT(b_cat_ID) AS total FROM books
-> GROUP BY b_cat_ID
-> HAVING total > 5
-> ORDER BY b_cat_ID;
```

```
+-----+-----+
| b_cat_ID | total |
+-----+-----+
	9
	6
	6
+-----+-----+
```

```
3 rows in set (0.00 sec)
```

- Запрос, извлекающий уникальные значения столбца *b\_cat\_ID*, большие двух:



- При этом в случае использования ключевого слова *WHERE* сначала производится выборка из таблицы с применением условия и лишь затем группировка результата, а в случае использования ключевого слова *HAVING* сначала происходит группировка таблицы и лишь затем выборка с применением условия. Допускается использование условия *HAVING* без группировки *GROUP BY*.

- **Использование функций.** Для решения специфических задач при выборке удобны встроенные функции MySQL. Большинство функций предназначено для использования в выражениях *SELECT* и *WHERE*. Существуют также специальные функции группировки для использования в выражении *GROUP BY* (см. выше).

- Каждая функция имеет уникальное имя и может иметь несколько аргументов (перечисляются через запятую в круглых скобках). Если аргументы отсутствуют, кр таблицам. Создание временных таблиц осуществляется при помощи оператора *CREATE temporary table*, синтаксис которого ничем не отличается от синтаксиса оператора *CREATE table*.

- Временная таблица автоматически удаляется по завершении соединения с сервером, а ее имя действительно только в течение данного соединения. Это означает, что два разных клиента могут использовать временные таблицы с одинаковыми именами без конфликта друг с другом или с существующей таблицей с тем же именем.

- 

- Пример выполнен других операторов и функций. Возможность выполнять математические расчеты на уровне *SELECT* позволяет проводить финансовый анализ значений таблиц и отображать полученные результаты в отчетах. Во всех выражениях MySQL (как в любом языке программирования) можно использовать скобки, чтобы контролировать порядок вычислений.

- **Операторы.** Под операторами подразумеваются конструкции языка, которые производят преобразование данных. Данные, над которыми совершается операция, называются операндами.

- В MySQL используются три типа операторов:

- арифметические операторы;
- операторы сравнения;
- логические операторы.

- *Арифметические операции.* В MySQL используются обычные арифметические операции: сложение (+), вычитание (-), умножение (\*), деление (/) и целочисленное деление *DIV* (деление и отсечение дробной части). Деление на 0 дает безопасный результат *NULL*.

- *Операторы сравнения.* При работе с операторами сравнения необходимо помнить о том, что, за исключением нескольких особо оговариваемых случаев, сравнение чего-либо со значением *NULL* дает в результате *NULL*. Это касается и сравнения значения *NULL* со значением *NULL*:

```
mysql> SELECT NULL=NULL;
+-----+
| NULL=NULL |
+-----+
|          1 |
+-----+
1 row in set (0.02 sec)
```

- 

- Корректнее использовать следующий запрос:

```
mysql> SELECT NULL IS NULL;
+-----+
| NULL IS NULL |
+-----+
|              1 |
+-----+
1 row in set (0.00 sec)
```

- 

- Поэтому следует быть предельно внимательными при работе с операторами сравнения, если операнды могут принимать значения

*NULL*.

- Наиболее часто используемые операторы сравнения приведены в табл. 7.

- *Логические операторы*. MySQL поддерживает все обычные логические операции, которые можно использовать в выражениях. Логические выражения в MySQL могут принимать значения 1 (истина), 0 (ложь) или *NULL*.

- Кроме того, следует учитывать, что MySQL интерпретирует любое ненулевое значение, отличное от *NULL*, как значение «истина». Основные логические операторы приведены в табл. 8.

- 
- **Практическая работа**
- При выполнении лабораторной работы необходимо:
  - для заданной предметной области построить два простых запроса на выборку с использованием операторов и функций MySQL;
  - составить отчет по лабораторной работе.

- Таблица 7

| Оператор                            | Значение                                                                                                                                                                                                                                  |
|-------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| =                                   | Оператор равенства. Возвращает 1 (истина), если операнды равны, и 0 (ложь), если не равны                                                                                                                                                 |
| <=>                                 | Оператор эквивалентности. Аналогичен обычному равенству, но возвращает только два значения: 1 (истина) и 0 (ложь). <i>NULL</i> не возвращает                                                                                              |
| <>                                  | Оператор неравенства. Возвращает 1 (истина), если операнды не равны, и 0 (ложь), если равны                                                                                                                                               |
| <                                   | Оператор «меньше». Возвращает 1 (истина), если левый операнд меньше правого, и 0 (ложь) – в противном случае                                                                                                                              |
| <=                                  | Оператор «меньше или равно». Возвращает 1 (истина), если левый операнд меньше правого или они равны, и 0 (ложь) – в противном случае                                                                                                      |
| >                                   | Оператор «больше». Возвращает 1 (истина), если левый операнд больше правого, и 0 (ложь) – в противном случае                                                                                                                              |
| >=                                  | Оператор «больше или равно». Возвращает 1 (истина), если левый операнд больше правого или они равны, и 0 (ложь) – в противном случае                                                                                                      |
| <i>n BETWEEN min AND max</i>        | Проверка диапазона. Возвращает 1 (истина), если проверяемое значение <i>n</i> находится между <i>min</i> и <i>max</i> , и 0 (ложь) – в противном случае                                                                                   |
| <i>IS NULL</i> и <i>IS NOT NULL</i> | Позволяют проверить, является ли значение значением <i>NULL</i> или нет                                                                                                                                                                   |
| <i>n IN (множество)</i>             | Принадлежность к множеству. Возвращает 1 (истина), если проверяемое значение <i>n</i> входит в список, и 0 (ложь) – в противном случае. В качестве множества может использоваться список литеральных значений или выражений или подзапрос |

- Таблица 8

| Оператор   | Пример         | Значение                                                                                                                                                                                                                                      |
|------------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>AND</i> | <i>n AND m</i> | Логическое <i>И</i> : истина <i>AND</i> истина = истина, ложь <i>AND</i> любое = ложь. Все остальные выражения оцениваются как <i>NULL</i>                                                                                                    |
| <i>OR</i>  | <i>n OR m</i>  | Логическое <i>ИЛИ</i> : истина <i>OR</i> любое = истина, <i>NULL OR</i> ложь = <i>NULL</i> , <i>NULL OR NULL</i> = <i>NULL</i> , ложь <i>OR</i> ложь = ложь                                                                                   |
| <i>NOT</i> | <i>NOT n</i>   | Логическое <i>НЕТ</i> : <i>NOT</i> истина = ложь, <i>NOT</i> ложь = истина. <i>NOT NULL</i> = <i>NULL</i>                                                                                                                                     |
| <i>XOR</i> | <i>n XOR m</i> | Логическое <i>исключающее ИЛИ</i> : истина <i>XOR</i> истина = ложь, истина <i>XOR</i> ложь = истина, ложь <i>XOR</i> истина = истина, ложь <i>XOR</i> ложь = ложь, <i>NULL XOR</i> любое = <i>NULL</i> , любое <i>XOR NULL</i> = <i>NULL</i> |

- **Переменные SQL и временные таблицы.** Часто результаты запроса необходимо использовать в последующих запросах. Для этого полученные данные необходимо сохранить во временных структурах. Эту задачу решают переменные SQL и временные таблицы. Объявление переменной начинается с символа *@*, за которым следует имя переменной. Значения переменным присваиваются посредством оператора *SELECT* с использованием оператора присваивания *:=*. Например:

```
mysql> SELECT @total := COUNT(*) FROM books;
+-----+
| @total := COUNT(*) |
+-----+
|                30 |
+-----+
1 row in set (0.42 sec)

mysql> SELECT @total;
+-----+
| @total |
+-----+
| 30     |
+-----+
1 row in set (0.02 sec)
```

- Объявляется переменная *@total*, которой присваивается число записей в таблице *books*. Затем в рамках текущего сеанса в последующих запросах появляется возможность использования данной переменной. Переменная действует только в рамках одного сеанса соединения с сервером MySQL и прекращает свое существование после разрыва соединения.
- Переменные также могут объявляться при помощи оператора *SET*:

```
mysql> SET @last=CURDATE()-INTERVAL 7 DAY;
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> SELECT CURDATE(), @last;
```

```
-----+-----+
| CURDATE() | @last |
-----+-----+
| 2009-12-22 | 2009-12-15 |
-----+-----+
1 row in set (0.00 sec)
```

- При использовании оператора *SET* в качестве оператора присваивания может выступать обычный знак равенства =. Оператор *SET* удобен тем, что он не возвращает результирующую таблицу. Не рекомендуется одновременно присваивать переменной некоторое значение и использовать эту переменную в одном запросе.

- Переменная SQL позволяет сохранить одно промежуточное значение. Когда необходимо сохранить результирующую таблицу, прибегают к временным таблицам. Создание временных таблиц осуществляется при помощи оператора *CREATE TEMPORARY TABLE*, синтаксис которого ничем не отличается от синтаксиса оператора *CREATE TABLE*.

- Временная таблица автоматически удаляется по завершении соединения с сервером, а ее имя действительно только в течение данного соединения. Это означает, что два разных клиента могут использовать временные таблицы с одинаковыми именами без конфликта друг с другом или с существующей таблицей с тем же именем.

- 

- **Пример выполнения работы**

- 1. Создадим простой запрос на выборку к таблице books, который выводит максимальную и минимальную цены товарных позиций, присваивая им соответственно псевдонимы maximum и minimum:

- 

- 2. Создадим простой запрос на выборку к таблице books, который выводит количество записей, соответствующих каждому из уникальных значений b\_cat\_ID. Для этого используем функцию count( ) вместе с выражением group by:

- 

- 

- **Лабораторная работа № 5**

Создание сложных запросов на выборку

- 

- Теоретические сведения

- Рассмотрим следующие вопросы:

- использование объединений в запросах к нескольким таблицам;

- создание вложенных запросов.

- В реальных приложениях часто требуется использовать сразу несколько таблиц БД. Запросы,



товарных позиций в каталогах:

- 
- Допустим, происходит расширение ассортимента и в списке каталогов появляется новый каталог «Компьютеры»:
- 
- Предыдущий запрос не отразит наличие нового каталога (таблица `books` не содержит записей, относящихся к новому каталогу). Выходом является использование левого объединения (таблица `catalogs` должна быть левой таблицей):
- 
- Пусть нужно вывести список покупателей и число осуществленных ими покупок, причем покупателей необходимо отсортировать по убыванию числа заказов:
- 
- В список не входят покупатели, которые не сделали ни одной покупки. Чтобы вывести полный список покупателей, необходимо вместо перекрестного объединения таблиц `users` и `orders` использовать левое объединение (левой таблицей должна быть таблица `users`):
- 
- Вложенный запрос. Позволяет использовать результат, возвращаемый одним запросом, в другом запросе. Так как результат возвращает только оператор `select`, то в качестве вложенного запроса всегда выступает `SELECT`-запрос. В качестве внешнего запроса может выступать запрос с участием любого SQL-оператора: `select`, `insert`, `update`, `delete`, `create table` и др.
- Пусть требуется вывести названия и цены товарных позиций из таблицы `books` для каталога «Базы данных» таблицы `catalogs`:
- 
- Получить аналогичный результат можно при помощи многотабличного запроса, но имеется ряд задач, которые решаются только при помощи вложенных запросов. Вложенный запрос может применяться не только с условием `WHERE`, но и в конструкциях `DISTINCT`, `GROUP BY`, `ORDER BY`, `LIMIT` и т. д. Различают:
  - вложенные запросы, возвращающие одно значение;
  - вложенные запросы, возвращающие несколько строк.
- В первом случае вложенный запрос возвращает скалярное значение или литерал, которое используется во внешнем запросе (подставляет результат на место своего выполнения). Например, необходимо определить название каталога, содержащего самую дорогую товарную позицию:
- 
- Наиболее часто вложенные запросы используются в операциях сравнения в условиях, которые задаются ключевыми словами `WHERE`, `HAVING` или `ON`.
- Однако следующий вложенный запрос вернет ошибку:

```
mysql> SELECT cat_name FROM catalogs
      -> WHERE cat_ID=(SELECT b_cat_ID FROM books);
• ERROR 1242 (21000): Subquery returns more than 1 row
```
- Чтобы выбрать строки из таблицы `catalogs`, у которых первичный ключ совпадает с одним из значений, возвращаемых вложенным запросом, следует воспользоваться конструкцией `IN`:

- ```
mysql> SELECT cat_name FROM catalogs
-> WHERE cat_ID IN (SELECT b_cat_ID FROM books GROUP BY b_cat_ID);
```

cat_name
Программирование
Интернет
Базы данных
Сети
Мультимедиа

```
5 rows in set (0.17 sec)
```

- Ключевое слово *ANY* может применяться с использованием любого оператора сравнения. Используется логика *ИЛИ*, т. е. достаточно, чтобы срабатывало хотя бы одно из многих условий. Запрос вида *WHERE X > ANY (SELECT Y...)* можно интерпретировать как «где *X* больше хотя бы одного выбранного *Y*». Соответственно, запрос вида *WHERE X < ANY (SELECT Y...)* интерпретируется как «где *X* меньше хотя бы одного выбранного *Y*». Рассмотрим запрос, возвращающий имена и фамилии покупателей, совершивших хотя бы одну покупку:

```
mysql> SELECT u_name, u_surname FROM users
-> WHERE user_ID=ANY(SELECT o_user_ID FROM orders);
```

u_name	u_surname
Александр	Иванов
Игорь	Симонов
Максим	Кузнецов
Александр	Корнеев

```
4 rows in set (0.03 sec)
```

- Ключевое слово *ALL* также может применяться с использованием любого оператора сравнения, но при этом используется логика *И*, то есть должны срабатывать все условия. Запрос вида *WHERE X > ALL (SELECT Y...)* интерпретируется как «где *X* больше любого выбранного *Y*». Соответственно, запрос вида *WHERE X < ALL (SELECT Y...)* интерпретируется как «где *X* меньше, чем все выбранные *Y*». Рассмотрим запрос, возвращающий все товарные позиции, цена которых превышает среднюю цену каждого из каталогов:

```
mysql> SELECT b_name, b_price FROM books
-> WHERE b_price>ALL(SELECT AVG(b_price) FROM books
-> GROUP BY b_cat_ID);
```

b_name	b_price
Visual FoxPro 9.0	660.00
Delphi. Полное руководство	500.00
Совершенный код	771.00
Принципы маршрутизации в Internet	428.00
Компьютерные сети	630.00
Сети. Поиск неисправностей	434.00
Безопасность сетей	462.00

```
7 rows in set (0.03 sec)
```

- Результирующая таблица, возвращаемая вложенным запросом, может не содержать ни одной строки. Для проверки этого факта могут использоваться ключевые слова *EXISTS* и *NOT EXISTS*.

- Запрос, формирующий список покупателей, совершивших хотя бы одну покупку, можно записать следующим образом:

```
mysql> SELECT u_name, u_surname FROM users
-> WHERE EXISTS (SELECT * FROM orders
-> WHERE orders.o_user_ID=users.user_ID);
```

u_name	u_surname
Александр	Иванов
Игорь	Симонов
Максим	Кузнецов
Александр	Корнеев

4 rows in set (0.00 sec)

- **Практическая работа**

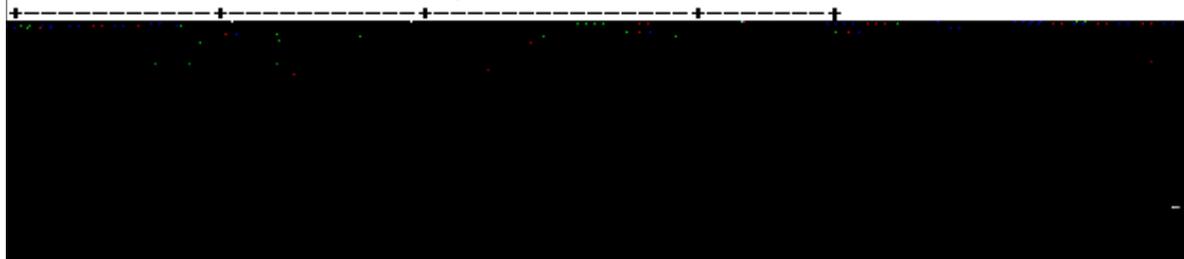
- При выполнении лабораторной работы необходимо:

- для заданной предметной области построить многотабличный запрос на выборку с использованием объединения;
- для заданной предметной области построить запрос на выборку, содержащий вложенный запрос;
- составить отчет по лабораторной работе.

- **Пример выполнения работы**

- 1. Создадим многотабличный запрос на выборку, который выводит фамилии, имена и отчества покупателей магазина, сделавших менее двух покупок:

```
mysql> SELECT users.u_surname,users.u_name,users.u_patronymic,
-> COUNT(orders.order_ID) AS total
-> FROM users LEFT JOIN orders ON users.user_ID=orders.o_user_ID
-> GROUP BY users.user_ID
-> HAVING total<2
-> ORDER BY total DESC;
```



- 2. Создадим запрос на выборку с вложенным запросом, выводящим перечень книг, которые не заказывались покупателями:

```
mysql> SELECT book_ID, b_name, b_price FROM books
-> WHERE NOT EXISTS (SELECT * FROM orders
-> WHERE orders.o_book_ID=books.book_ID);
```

book_ID	b_name	b_price
1	JavaScript в кармане	42.00
2	Visual FoxPro 9.0	660.00
3	C++ Как он есть	218.00
4	Создание приложений с помощью С#	169.00
5	Delphi. Народные советы	243.00
6	Delphi. Полное руководство	500.00
7	Профессиональное программирование на PHP	309.00
9	Практика программирования	214.00
11	Поиск в Internet	107.00
12	Web-конструирование	177.00
13	Самоучитель Интернет	121.00
14	Популярные интернет-браузеры	82.00
15	Общение в Интернете	85.00
16	Базы данных	326.00
17	Базы данных. Разработка приложений	189.00
18	Раскрытие тайн SQL	200.00
19	Практикум по Access	87.00
21	Сети. Поиск неисправностей	434.00
22	Безопасность сетей	462.00
23	Анализ и диагностика компьютерных сетей	344.00
24	Локальные вычислительные сети	82.00
25	Цифровая фотография	149.00
26	Музыкальный компьютер для гитариста	217.00
27	Видео на ПК	231.00
28	Мультипликация во Flash	211.00
29	Запись CD и DVD	167.00
30	Запись и обработка звука на компьютере	51.00

• 27 rows in set (0.03 sec)

## • Лабораторная работа № 6

### Создание хранимых процедур

- 
- **Теоретические сведения**
- На практике часто требуется повторять последовательность одинаковых запросов. Хранимые процедуры позволяют объединить последовательность таких запросов и сохранить их на сервере. После этого клиентам достаточно послать один запрос на выполнение хранимой процедуры.
- Хранимые процедуры обладают следующими преимуществами.
  - *Повторное использование кода* – после создания хранимой процедуры ее можно вызывать из любых приложений и SQL-запросов.
  - *Сокращение сетевого трафика* – вместо нескольких запросов экономнее послать серверу запрос на выполнение хранимой процедуры и сразу получить ответ.
  - *Безопасность* – действия не приведут к нарушению целостности данных, т.к. для выполнения хранимой процедуры пользователь должен иметь привилегию.
  - *Простота доступа* – хранимые процедуры позволяют инкапсулировать сложный код и оформить его в виде простого вызова.
  - *Выполнение бизнес-логики* – хранимые процедуры позволяют перенести код сохранения целостности БД из прикладной программы на сервер БД. Бизнес-логика в виде хранимых процедур не зависит от языка разработки приложения.
- **Создание хранимых процедур.** Реализуется оператором
- *CREATE PROCEDURE имя\_процедуры ( [ параметр [ , ... ] ] )*
- *[характеристика ...] тело\_процедуры*
- В скобках передается необязательный список параметров,

перечисленных через запятую. Каждый параметр позволяет передать в процедуру (из процедуры) входные данные (результат работы) и имеет следующий синтаксис:

- `[ IN | OUT | INOUT ] имя_параметра тип`

- Ключевые слова *IN*, *OUT*, *INOUT* задают направление передачи данных:

- *IN* – данные передаются строго внутрь хранимой процедуры; если параметру с данным модификатором присваивается новое значение, при выходе из процедуры оно не сохраняется и параметр принимает значение, которое он имел до вызова;

- *OUT* – данные передаются строго из хранимой процедуры, если параметр имеет какое-то начальное значение, то внутри хранимой процедуры это значение во внимание не принимается;

- *INOUT* – значение этого параметра как принимается во внимание внутри процедуры, так и сохраняет свое значение при выходе из нее.

- Список аргументов, заключенных в круглые скобки, присутствует всегда. Если аргументы отсутствуют, следует использовать пустой список. Если ни один из модификаторов не указан, считается, что параметр объявлен с ключевым словом *IN*.

- Телом процедуры является составной оператор *BEGIN ... END*, внутри которого могут располагаться другие операторы:

- `[ label: ] BEGIN`

- `statements`

- `END [ label ]`

- Оператор, начинающийся с необязательной метки *label* (любое уникальное имя), может заканчиваться выражением *END label*. Внутри составного оператора *BEGIN ... END* может находиться другой составной оператор. Если хранимая процедура содержит один запрос, то составной оператор можно не использовать.

- При работе с хранимыми процедурами символ точки с запятой в конце запроса воспринимается консольным клиентом как сигнал к отправке запроса на сервер. Поэтому следует переопределить разделитель запросов – например, вместо точки с запятой использовать последовательность `//` :

```
mysql> DELIMITER //
mysql> SELECT VERSION< >//
+-----+
| VERSION< > |
+-----+
| 5.0.51b-community-nt |
+-----+
1 row in set (0.00 sec)
```

- 

- Пример создания простейшей хранимой процедуры:

```
mysql> CREATE PROCEDURE my_version()
-> BEGIN
-> SELECT VERSION();
-> END //
Query OK, 0 rows affected (0.00 sec)
```

- Чтобы вызвать хранимую процедуру, необходимо применить оператор *CALL*, после которого помещается имя процедуры и ее параметры в круглых скобках:

```
mysql> CALL my_version()//
+-----+
| VERSION() |
+-----+
| 5.0.51b-community-nt |
+-----+
1 row in set (0.00 sec)
```

```
Query OK, 0 rows affected (0.00 sec)
```

- Рекомендуется избегать использования названий хранимых процедур, совпадающих с именами встроенных функций MySQL. В теле хранимой процедуры можно использовать многострочный комментарий, который начинается с последовательности */\** и заканчивается последовательностью *\*/*.

- Рассмотрим хранимые процедуры с параметрами. Создадим и вызовем процедуру, которая присваивает пользовательской переменной *@x* новое значение:

```
mysql> CREATE PROCEDURE set_x(IN value INT)
-> BEGIN
-> SET @x = value;
-> END//
Query OK, 0 rows affected (0.00 sec)
```

```
mysql>
mysql> CALL set_x(123456)//
Query OK, 0 rows affected (0.00 sec)
```

- Через параметр *value* процедуре передается числовое значение 123456, которое она присваивает пользовательской переменной *@x*. Модификатор *IN* сообщает, что данные передаются внутрь функции. Проверим корректность работы процедуры:

```
mysql> SELECT @x//
+-----+
| @x |
+-----+
| 123456 |
+-----+
1 row in set (0.00 sec)
```

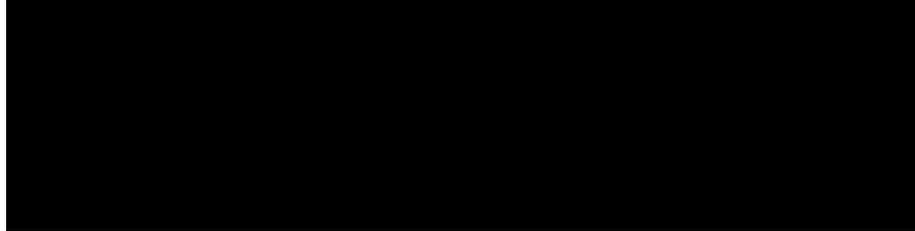
- В отличие от пользовательской переменной *@x*, которая является глобальной и доступна как внутри хранимой процедуры *set\_x ( )*, так и вне ее, параметры процедуры являются локальными и доступны для использования только внутри нее.

- Создадим процедуру *numcatalogs()*, которая подсчитывает

число записей в таблице *catalogs* базы данных *book*:

```
mysql> CREATE PROCEDURE numcatalogs(OUT total INT)
-> BEGIN
-> SELECT COUNT(*) INTO total FROM catalogs;
-> END //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL numcatalogs(@a)//
Query OK, 0 rows affected (0.01 sec)
```



- Хранимая процедура *numcatalogs()* имеет один целочисленный параметр *total*, в котором сохраняется число записей в таблице *catalogs*. Осуществляется это при помощи оператора *SELECT ... INTO ... FROM*. В качестве параметра функции *numcatalogs()* передается пользовательская переменная *@a*.

- Создадим хранимую процедуру *catalogname()*, которая будет возвращать по первичному ключу *catID* название каталога *cat\_name*. Для этого потребуется определить параметр *id* с атрибутом *IN*, и *catalog* с атрибутом *OUT*.

```
mysql> CREATE PROCEDURE catalogname(IN id INT, OUT catalog TINYTEXT)
-> BEGIN
-> SELECT cat_name INTO catalog FROM catalogs
-> WHERE catID = id;
-> END //
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SET @id = 5//
Query OK, 0 rows affected (0.00 sec)

mysql> CALL catalogname(@id, @name)//
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> SELECT @id, @name//
+-----+-----+
| @id | @name          |
+-----+-----+
| 5   | Мультимедиа   |
+-----+-----+
1 row in set (0.00 sec)
```

- **Операторы управления потоком данных.** Хранимые процедуры позволяют реализовать сложную логику с помощью операторов ветвления и циклов. Вне хранимых процедур эти операторы применять нельзя. Ветвление программы по условию позволяет реализовать оператор:

- *IF* лог\_выражение *THEN* оператор
- *[ELSEIF* лог\_выражение *THEN* оператор] ...
- *[ELSE* оператор]
- *END IF* ;
- Логическое выражение может принимать два значения:

- 0 (ложь);
- значение, отличное от нуля (истина).
  - Если логическое выражение истинно, то выполняется оператор в блоке *THEN*, иначе выполняется список операторов в блоке *ELSE* (если блок *ELSE* имеется). В логических выражениях можно использовать операторы сравнения ( = , > , >= , <> , < , <= ). Логические выражения можно комбинировать с помощью операторов && (И), а также || (ИЛИ). Если в блоках *IF*, *ELSEIF* и *ELSE* – два или более операторов, необходимо использовать составной оператор *BEGIN ... END*.
    - Множественный выбор позволяет осуществить оператор:
      - *CASE* выражение
      - *WHEN* значение *THEN* оператор
      - [*WHEN* значение *THEN* оператор] ...
      - [*ELSE* оператор]
      - *END CASE* ;
    - Выражение сравнивается со значениями. Как только найдено соответствие, выполняется соответствующий оператор. Если соответствия не найдены, выполняется оператор, размещенный после ключевого слова *ELSE* (если оно присутствует).
      - В MySQL имеется несколько операторов, позволяющих реализовать циклы. Первый оператор цикла имеет следующий синтаксис:
        - [*label*:] *WHILE* условие *DO*
        - операторы
        - *END WHILE* [*label*] ;
      - Операторы выполняются в цикле, пока истинно условие. При каждой итерации условие проверяется, и если при очередной проверке оно будет ложным (0), цикл завершится. Если условие ложно с самого начала, то цикл не выполнится ни разу. Если в цикле выполняется более одного оператора, не обязательно заключать их в блок *BEGIN ... END*, т. к. эту функцию выполняет сам оператор *WHILE*.
        - Досрочный выход из цикла обеспечивает оператор:
          - *LEAVE label* ;
        - Оператор прекращает выполнение блока, помеченного меткой *label* (например, прекращает выполнение цикла по достижении критического числа итераций).
          - Досрочное прекращение цикла также обеспечивает оператор
            - *ITERATE label* ;
  - В отличие от оператора *LEAVE* оператор *ITERATE* не

прекращает выполнение цикла, он лишь выполняет досрочное прекращение текущей итерации. Оператор *LEAVE* эквивалентен оператору *BREAK*, а оператор *ITERATE* эквивалентен оператору *CONTINUE* в С-подобных языках программирования.

- 
- Второй оператор цикла имеет следующий вид:
- *[ label: ] REPEAT*
- операторы *UNTIL* условие *END REPEAT [label] ;*
- Условие проверяется не в начале, а в конце оператора цикла.

Таким образом, цикл выполняет по крайней мере одну итерацию независимо от условия. Следует отметить, что цикл выполняется, пока условие ложно. Оператор цикла может быть снабжен необязательной меткой *label*, по которой можно осуществлять досрочный выход из цикла при помощи операторов *LEAVE* и *ITERATE*.

- Реализовать бесконечный цикл позволяет оператор
- *[ label: ] LOOP*
- операторы *END LOOP [ label ] ;*

• Цикл *LOOP* (в отличие от операторов *WHILE* и *REPEAT*) не имеет условий выхода. Поэтому данный цикл должен обязательно иметь в составе оператор *LEAVE*.

- Осуществлять безусловный переход позволяет оператор
- *GOTO label ;*
- Оператор осуществляет переход к оператору, помеченному меткой *label*. Это может быть как оператор *BEGIN*, так и любой из циклов: *WHILE*, *REPEAT* и *LOOP*. Кроме того, метка может быть не привязана ни к одному из операторов, а объявлена при помощи оператора

- *LABEL label ;*

• Использовать оператор *GOTO* для реализации циклов не рекомендуется, т. к. обычные циклы гораздо нагляднее и проще поддаются модификации, в них сложнее допустить логическую ошибку.

• **Удаление хранимых процедур.** Для удаления процедур используется оператор

- *DROP PROCEDURE [IF EXISTS] имя\_процедуры ;*

• Если удаляемой процедуры с таким именем не существует, оператор возвращает ошибку, которую можно подавить, если использовать необязательное ключевое слово *IF EXISTS*.

• **Обработчики ошибок.** При выполнении хранимых процедур могут возникать ошибки. MySQL позволяет каждой возникающей в

хранимой процедуре ошибке назначить свой обработчик, который в зависимости от ситуации и серьезности ошибки может как прекратить, так и продолжить выполнение процедуры.

- Для объявления такого обработчика предназначен оператор
- *DECLARE* *именованный\_обработчик* *HANDLER FOR код\_ошибки* [*, ...*] *выражение*;
- Выражение содержит SQL-запрос, который выполняется при срабатывании обработчика. Тип обработчика может принимать одно из трех значений:
  - *CONTINUE* – выполнение текущей операции продолжается после выполнения оператора обработчика;
  - *EXIT* – выполнение составного оператора *BEGIN ... END*, в котором объявлен обработчик, прекращается;
  - *UNDO* – данный вид обработчика в текущей версии не поддерживается.
- Обработчик может быть привязан сразу к нескольким ошибкам, для этого их коды следует перечислить через запятую. Код ошибки, для которой будет происходить срабатывание обработчика, может принимать следующие значения:
  - *SQLSTATE* [*VALUE*] *значение* – значение *SQLSTATE* является пятисимвольным кодом ошибки в шестнадцатеричном формате (стандарт в SQL); примеры кодов – 'HY000', 'HY001', '42000' и т. д.; один код обозначает сразу несколько ошибок;
  - *SQLWARNING* – любое предупреждение MySQL; это ключевое слово позволяет назначить обработчик для всех предупреждений; обрабатываются любые события, для которых код *SQLSTATE* начинается с 01;
  - *NOT FOUND* – любая ошибка, связанная с невозможностью найти объект (таблицу, процедуру, функцию, столбец и т. п.); обрабатываются любые события, для которых код *SQLSTATE* начинается с 02;
  - *SQLException* – ошибки, не охваченные ключевыми словами *SQLWARNING* и *NOT FOUND*;
  - *mysql\_error\_code* – обычные четырехзначные ошибки MySQL, такие как 1020, 1232, 1324 и т. п.;
  - именованное условие (см. ниже).
    - При указании кода ошибки можно использовать не только целочисленные коды, но и именованные условия, которые объявляются при помощи оператора
      - *DECLARE* *именованное\_условие* *CONDITION FOR код\_ошибки*;
      - Оператор объявляет именованное условие для кода ошибки. Так, для обрабатываемой ошибки 1062 (23000) – дублирование уникального индекса, оператор может выглядеть следующим образом:
        - *DECLARE* 'violation' *CONDITION FOR SQLSTATE* '23000';
        - *DECLARE* 'violation' *CONDITION FOR* 1062;
        - Первое объявление охватывает все ошибки со статусом 23000,

второй вид ошибок более узкий и включает только дублирование уникального индекса.

- **Курсоры.** Если результирующий запрос возвращает одну запись, поместить результаты в промежуточные переменные можно с помощью оператора *SELECT ... INTO ... FROM*. Однако результирующие таблицы чаще содержат несколько записей, и использование такой конструкции приводит к возникновению ошибки 1172: «Результат содержит более чем одну строку».

- Избежать ошибки можно, добавив предложение *LIMIT 1* или назначив *CONTINUE*-обработчик ошибок. Однако такая процедура реализует не то поведение, которое ожидает пользователь. Кроме того, существуют ситуации, когда требуется обработать именно многострочную результирующую таблицу.

- Например, пусть требуется вернуть записи одной таблицы, отвечающие определенному условию, и на основании этих записей создать новую таблицу. Решить эту задачу можно с помощью курсоров, которые позволяют в цикле просмотреть каждую строку результирующей таблицы запросов. Работа с курсорами похожа на работу с файлами – сначала открытие курсора, затем чтение и после закрытие.

- Работа с курсорами происходит по следующему алгоритму:

- 1. При помощи инструкции *DECLARE курсор CURSOR FOR* связывается имя курсора с выполняемым запросом.

- 2. Оператор *OPEN* выполняет запрос, связанный с курсором, и устанавливает курсор перед первой записью результирующей таблицы.

- 3. Оператор *FETCH* помещает курсор на первую запись результирующей таблицы и извлекает данные из записи в локальные переменные хранимой процедуры. Повторный вызов оператора *FETCH* приводит к перемещению курсора к следующей записи, и так до тех пор, пока записи в результирующей таблице не будут исчерпаны. Эту операцию удобно осуществлять в цикле.

- 4. Оператор *CLOSE* прекращает доступ к результирующей таблице и ликвидирует связь между курсором и результирующей таблицей.

- 

- **Практическая работа**

- При выполнении лабораторной работы необходимо:

- для заданной предметной области написать две хранимые процедуры и включить их в БД;
- составить отчет по лабораторной работе.

- 

- **Пример выполнения работы**

- 1. Создадим хранимую процедуру, которая выводит число заказов покупателя по вводимому в качестве параметра процедуры

коду покупателя.

```
mysql> CREATE PROCEDURE num(OUT total INT, IN user_kod INT)
-> BEGIN
-> SELECT COUNT(*) INTO total FROM orders WHERE o_user_ID=user_kod;
-> END
-> //
Query OK, 0 rows affected (0.00 sec)
```

- Параметр *total* является выходным, его значение равно числу заказов покупателя, код которого записывается во входной параметр *user\_kod*. Процедура считает все строки, где код клиента совпадает с параметром *user\_kod*.

- До вызова процедуры присваиваем параметру процедуры значение кода клиента. Затем вызываем процедуру оператором *CALL*. Для вывода результата можно воспользоваться оператором *SELECT*.

```
mysql> SET @user_kod=3//
Query OK, 0 rows affected (0.00 sec)

mysql> CALL num(@total,@user_kod)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @total,@user_kod//
+-----+-----+
| @total | @user_kod |
+-----+-----+
| 2      | 3          |
+-----+-----+
1 row in set (0.00 sec)
```

- 2. Создадим хранимую процедуру, которая записывает в новую таблицу *fevral* все заказы, сделанные в феврале 2009 г. Предварительно необходимо создать новую пустую таблицу *fevral* со структурой, аналогичной структуре таблицы *orders*.

```
mysql> CREATE TABLE fevral(
-> f_order_ID int(6) NOT NULL,
-> f_o_user_ID int NOT NULL,
-> f_o_book_ID int NOT NULL,
-> f_o_time datetime NOT NULL,
-> f_o_number int(6) NOT NULL,
-> PRIMARY KEY (f_order_ID)
-> >TYPE=InnoDB//
Query OK, 0 rows affected, 1 warning (0.08 sec)
```

- Хранимая процедура *ord\_fevr* ( ) использует курсор *curf*, который в цикле читает данные из таблицы *orders* и добавляет их в таблица одного из запросов происходит сбой, это может нарушить **ЦЕЛОСТНОСТЬ** БД (товар может быть продан, а число товарных позиций на складе не обновлено). **Чтобы сохранить целостность БД, все изменения должны выполняться как единое целое. Либо все изменения успешно выполняются, либо, в случае сбоя, БД принимает состояние, КОТОРОЕ было до начала изменений. Это обеспечивается СРЕДСТВАми обработки транзакций.**

- Транзакция – последовательность операторов SQL, выполняющихся как единая операция, которая не прерывается друзанных с изменением содержимого таблицы, к которым можно привязать триггер:

- *INSERT* – вставка новых данных в таблицу;

- *DELETE* – удаление данных из таблицы;
  - *UPDATE* – обновление данных в таблице.
- Например, при оформлении нового заказа, т. е. при добавлении новой записи в таблицу *orders*, можно создать триггер, автоматически вычитающий число заказанных товарных позиций в таблице *books*.
    - **Создание триггеров**
    - Создать новый триггер позволяет оператор:
      - *CREATE TRIGGER trigger\_name trigger\_time trigger\_event*
      - *ON tbl\_name FOR EACH ROW trigger\_stmt ;*
    - Оператор создает триггер с именем *trigger\_name*, привязанный к таблице *tbl\_name*. Не допускается привязка триггера к временной таблице или представлению. Конструкция *trigger\_time* указывает момент выполнения триггера и может принимать два значения:
      - *BEFORE* – действия триггера производятся до выполнения операции изменения таблицы;
      - *AFTER* – действия триггера производятся после выполнения операции изменения таблицы.
        - Конструкция *trigger\_event* показывает, на какое событие должен реагировать триггер, и может принимать три значения:
          - *INSERT* – триггер привязан к событию вставки новой записи в таблицу;
          - *UPDATE* – триггер привязан к событию обновления записи таблицы;
          - *DELETE* – триггер привязан к событию удаления записей таблицы.
        - Для таблицы *tbl\_name* может быть создан только один триггер для каждого из событий *trigger\_event* и момента *trigger\_time*. Таким образом, для каждой из таблиц может быть создано всего шесть триггеров.
        - Конструкция *trigger\_stmt* представляет тело триггера – оператор, который необходимо выполнить при возникновении события *trigger\_event* в таблице *tbl\_name*.
          - Если требуется выполнить несколько операторов, то необходимо использовать составной оператор *BEGIN ... END*. Синтаксис и допустимые операторы такие же, как и у хранимых процедур. Внутри составного оператора *BEGIN ... END* допускаются все специфичные для хранимых процедур операторы и конструкции:
            - другие составные операторы *BEGIN ... END*;
            - операторы управления потоком (*IF, CASE, WHILE, LOOP, REPEAT, LEAVE, ITERATE*);
            - объявления локальных переменных при помощи оператора *DECLARE* и назначение им значений при помощи оператора *SET*;
            - именованные условия и обработчики ошибок.
          - В MySQL триггеры нельзя привязать к каскадному обновлению или удалению записей из таблицы типа *InnoDB* по связи

первичный ключ/внешний ключ.

- Триггеры сложно использовать, не имея доступа к новым записям, которые вставляются в таблицу, или старым записям, которые обновляются или удаляются. Для доступа к новым и старым записям используются префиксы *NEW* и *OLD* соответственно. Если в таблице обновляется поле *total*, то получить доступ к старому значению можно по имени *OLD.total*, а к новому – *NEW.total*.

- Пример простейшего триггера для учебной БД *book* см. в пункте «Пример выполнения работы» (пример 1). Он демонстрирует работу триггеров после добавления записи в таблицу без вмешательства в запрос. Рассмотрим триггер, который будет включаться до вставки новых записей в таблицу *orders* и ограничивает число заказываемых товаров до 1:

```
mysql> CREATE TRIGGER restrict_count BEFORE INSERT ON orders
-> FOR EACH ROW
-> BEGIN
-> SET NEW.o_number=1;
```

```
mysql> INSERT INTO orders VALUES (NULL,1,2,NOW(),10)//
Query OK, 1 row affected (0.01 sec)
```

```
mysql> SELECT * FROM orders WHERE orderID = LAST_INSERT_ID();//
```

orderID	o_userID	o_bookID	o_time	o_number
16	1	2	2009-10-23 20:26:19	1

```
1 row in set (0.00 sec)
```

- Часто при обновлении полей таблицы производится попытка добавления некорректных значений. Пример триггера, который при добавлении нового покупателя преобразует полные имена и отчества в инициалы, см. в пункте «Пример выполнения работы» (пример 2). Он привязан к событию *INSERT*. Чтобы имя и отчество не могло быть отредактировано при помощи оператора *UPDATE*, можно создать триггер, привязанный к событию *UPDATE*.

- **Удаление триггеров.** Удалить существующий триггер позволяет оператор

- *DROP TRIGGER trigger\_name;*

## • Практическая работа

- При выполнении лабораторной работы необходимо:

- для заданной предметной области написать два триггера для разных таблиц базы данных;
- составить отчет по лабораторной работе.

## • Пример выполнения работы

- 1. Создадим триггер, который при оформлении нового заказа

(при добавлении новой записи в таблицу *orders*) будет увеличивать на 1 значение пользовательской переменной *@tot*.

```
mysql> delimiter //
mysql> CREATE TRIGGER sub_count AFTER INSERT ON orders
-> FOR EACH ROW
-> BEGIN
-> SET @tot =@tot+1;
-> END//
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT @tot //
+-----+
| @tot |
+-----+
| NULL |
+-----+
1 row in set (0.00 sec)
```

- Для корректной работы триггера необходимо, чтобы пользовательская переменная *@tot* имела значение, отличное от *NULL*, т. к. операция сложения с *NULL* также приводит к *NULL*.

```
mysql> SET @tot=5//
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO orders VALUES (NULL,1,5,NOW(),10)//
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @tot//
+-----+
| @tot |
+-----+
| 6    |
+-----+
1 row in set (0.00 sec)
```

- 2. Создадим триггер, который при добавлении новых покупателей преобразует имена и отчества покупателей в инициалы.

```
mysql> CREATE TRIGGER restrict_user BEFORE INSERT ON users
-> FOR EACH ROW
-> BEGIN
-> SET NEW.u_name = LEFT(NEW.u_name,1);
-> SET NEW.u_patronymic = LEFT(NEW.u_patronymic,1);
-> END//
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> INSERT INTO users VALUES (NULL, 'Светлана', 'Петровна', 'Титов',
-> '83-89-00', NULL, 'active')//
Query OK, 1 row affected (0.03 sec)
```

```
mysql> SELECT u_surname, u_name, u_patronymic FROM users
-> WHERE userID = LAST_INSERT_ID()//
+-----+-----+-----+
| u_surname | u_name | u_patronymic |
+-----+-----+-----+
| Титова   | С     | П             |
+-----+-----+-----+
1 row in set (0.00 sec)
```

- 
- Лабораторная работа № 8

## Транзакции

- 
- **Теоретические сведения**
- Изменения БД часто требуют выполнения нескольких запросов, например при покупке в электронном магазине требуется добавить запись в таблицу заказов и уменьшить число товарных позиций на складе. В промышленных БД одно событие может затрагивать большее число таблиц и требовать многочисленных запросов.
- Если на этапе выполнения одного из запросов происходит сбой, это может нарушить целостность БД (товар может быть продан, а число товарных позиций на складе не обновлено). Чтобы сохранить целостность БД, все изменения должны выполняться как единое целое. Либо все изменения успешно выполняются, либо, в случае сбоя, БД принимает состояние, которое было до начала изменений. Это обеспечивается средствами обработки транзакций.
- *Транзакция* – последовательность операторов SQL, выполняющихся как единая операция, которая не прерывается другими клиентами. Пока происходит работа с записями таблицы (обновление или удаление), никто другой не может получить доступ к этим записям, т. к. MySQL автоматически блокирует доступ к ним.
- Таблицы *ISAM*, *MyISAM* и *HEAP* не поддерживают транзакции. В настоящий момент их поддержка осуществляется только в таблицах *BDB* и *InnoDB*.
- Транзакции позволяют объединять операторы в группу и гарантировать, что все операторы группы будут выполнены успешно. Если часть транзакции выполняется со сбоем, результаты выполнения всех операторов транзакции до места сбоя отменяются, приводя БД к виду, в котором она была до выполнения транзакции.
- По умолчанию MySQL работает в режиме автоматического завершения транзакций, т. е. как только выполняется оператор обновления данных, который модифицирует таблицу, изменения тут же сохраняются на диске. Чтобы объединить операторы в транзакцию, следует отключить этот режим: *SET AUTOCOMMIT=0*;
- После отключения режима для завершения транзакции необходимо ввести оператор *COMMIT*, для отката – *ROLLBACK*.
- Включить режим автоматического завершения транзакций для отдельной последовательности операторов можно оператором *START TRANSACTION*.
- Для таблиц *InnoDB* есть операторы *SAVEPOINT* и *ROLLBACK TO SAVEPOINT*, которые позволяют работать с именованными точками начала транзакции.

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO catalogs VALUES(NULL,'Периферия');
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT point1;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO catalogs VALUES(NULL,'Разное');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM catalogs;
+-----+-----+
| cat_ID | cat_name      |
+-----+-----+
|      1 | Программирование |
|      2 | Интернет      |
|      3 | Базы данных    |
|      4 | Сети           |
|      5 | Мультимедиа   |
|     12 | Периферия     |
|     13 | Разное        |
+-----+-----+
7 rows in set (0.00 sec)
```

```
mysql> ROLLBACK TO SAVEPOINT point1;
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> SELECT * FROM catalogs;
+-----+-----+
| cat_ID | cat_name      |
+-----+-----+
|      1 | Программирование |
|      2 | Интернет      |
|      3 | Базы данных    |
|      4 | Сети           |
|      5 | Мультимедиа   |
|     12 | Периферия     |
+-----+-----+
6 rows in set (0.00 sec)
```

- Оператор *SAVEPOINT* устанавливает именованную точку начала транзакции с именем *point1*. Оператор *ROLLBACK TO SAVEPOINT point1* откатывает транзакцию к состоянию, в котором находилась БД на момент установки именованной точки. Все точки сохранения транзакций удаляются, если выполняются операторы *COMMIT* или *ROLLBACK* без указания имени точки сохранения.

### • Практическая работа

- При выполнении лабораторной работы необходимо:
  - создать транзакцию, произвести ее откат и фиксацию;
  - составить отчет по лабораторной работе.

#### • Пример выполнения работы

• Для выполнения задания объединим несколько операций по добавлению в таблицу *catalogs* новых каталогов, а затем произведем откат транзакции, т. е. отмену произведенных действий. Отключаем режим автоматическим уровнем привилегий. Учетной записи редактора можно предоставить привилегии на просмотр таблицы, добавление новых *записей* и обновление уже существующих. Администратору БД можно предоставить более широкие полномочия (**возможность создания таблиц, редактирования и удаления уже существующих**). Для пользователя БД достаточно лишь просмотра таблиц.

- Рассмотрим следующие *вопросы*:

- создание, редактирование и удаление учетных записей пользователей;
- назначение и отмена привилегий.
- Учетная запись является составной и яет еще один способ организации данных.

Представление – это запрос на выборку, которому присваивается уникальное имя и который можно сохранять или удалять из БД как хранимую процедуру. Представления позволяют

увидеть результаты сохраненного запроса так, как будто это полноценная таблица. MySQL, встретив в запросе ссылку на представление, ищет его определение в БД. Пользовательский запрос с участием представления преобразуется в эквивалентный запрос к исходным таблицам. Если определение представления простое, то каждая строка представления формируется «на лету». Если определение сложное, MySQL материализует представление в виде временной таблицы. Клиент, обращаясь к представлению, будет видеть только столбцы результирующей таблицы. Не имеет значения, сколько столбцов в исходной таблице и является ли запрос, лежащий в основе представления, одно- или многотабличным. Клиенту можно запретить обращаться к исходным таблицам, но снабдить привилегиями обращения к представлениям. На одном наборе таблиц можно создать гибкие системы доступа.

- Преимущества представлений:
  - *безопасность* – каждый пользователь имеет доступ к небольшому числу представлений, содержащих только ту информацию, которую ему позволено знать;
  - *простота запросов* – с помощью представления можно извлечь данные из нескольких таблиц и представить их как одну таблицу (запрос ко многим таблицам заменяется однотоабличным запросом к представлению);
  - *простота структуры* – представления позволяют создать для каждого пользователя собственную структуру БД (отображаются данные, которые ему нужны);
  - *защита от изменений* – таблицы и их структура могут постоянно изменяться и переименовываться; представления позволяют создавать виртуальные таблицы со старыми именами и структурой, позволяя избежать модификации приложений.
- Недостатки представлений:
  - *производительность* – представления создают видимость существования таблицы, и MySQL приходится преобразовывать запрос к представлению в запрос к исходным таблицам; если представление отображает многотабличный запрос, то простой запрос к представлению становится сложным объединением;
  - *ограничение на обновление* – когда пользователь пытается обновить строки представления, MySQL необходимо обновить строки в исходных таблицах; это возможно только для простых представлений, сложные представления доступны только для выборки.
- Поэтому не стоит везде применять представления вместо исходных таблиц.
  - **Создание представлений.** Осуществляется при помощи оператора
- `CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED / MERGE / TEMPTABLE}] VIEW view_name [(column_list)] AS select_statement`
- `[WITH [CASCADED / LOCAL] CHECK OPTION];`
  - Оператор создает представление *view\_name* со столбцами, перечисленными в *column\_list*, на основании запроса *select\_statement*. Рассмотрим создание представления *cat*, которое дублирует таблицу *catalogs* базы данных *book*:

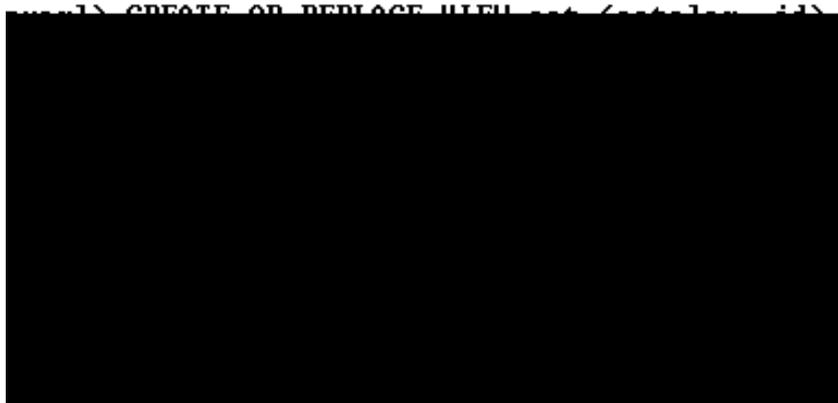
```
mysql> CREATE VIEW cat AS SELECT * FROM catalogs;
Query OK, 0 rows affected (0.02 sec)

mysql> SELECT * FROM cat;
+-----+-----+
| catID | cat_name |
+-----+-----+
| 1     | Программирование |
| 2     | Интернет |
| 3     | Базы данных |
| 4     | Сети |
| 5     | Мультимедиа |
+-----+-----+
5 rows in set (0.00 sec)
```

- Представление рассматривается как полноценная таблица и может быть просмотрено в списке таблиц БД при помощи оператора *SHOW TABLES*:

```
mysql> SHOW TABLES;
+-----+
| Tables_in_book |
+-----+
| books |
| cat |
| catalogs |
| orders |
| users |
+-----+
5 rows in set (0.00 sec)
```

- При создании представления можно явно указать список столбцов, изменить их названия и порядок следования, например:



- Представления, не содержащие дополнительных столбцов, называются *вертикальными представлениями*. Они применяются для ограничения доступа пользователей к столбцам таблицы. Пример вертикального представления см. ниже.

- Кроме вертикальных представлений используются *горизонтальные представления*, которые делают видимыми только те строки, с которыми работают пользователи. Например, чтобы в электронном магазине каждый менеджер видел только те товарные позиции, за которые отвечает, можно создать представления для менеджеров. Учетные записи менеджеров следует лишить привилегии доступа к таблице и разрешить просматривать только свои представления.

- Создадим представление *manager1* для менеджера, работающего с каталогами «Интернет» и «Сети»:

```
mysql> CREATE VIEW manager1
-> AS SELECT * FROM books
-> WHERE b_catID IN (SELECT catID
-> FROM catalogs
-> WHERE cat_name = 'Интернет' OR cat_name = 'Сети')
-> ORDER BY b_name;
Query OK, 0 rows affected (0.01 sec)
```

- Наиболее удобно использовать представления для формирования сгруппированных таблиц. При работе с такими таблицами MySQL самостоятельно формирует временную таблицу – см. пункт «Пример выполнения работы» (пример 2).

- **Удаление представлений.** Выполняется с помощью оператора:

- *DROP VIEW [IF EXISTS] view\_name [, view\_name] ... ;*

- Оператор позволяет уничтожить одно или несколько представлений, например:

```
mysql> DROP VIEW cat, list_user, price;
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> SHOW TABLES;
```

```
+-----+
| Tables_in_book |
+-----+
| books          |
| catalogs       |
| orders         |
| users          |
+-----+
```

```
4 rows in set (0.00 sec)
```

- **Практическая работа**

- При выполнении лабораторной работы необходимо:

- для заданной предметной области создать два представления в БД;
- сформировать запрос к одному из представлений;
- составить отчет по лабораторной работе.

- **Пример выполнения работы**

- Создадим вертикальное представление *list\_user*, которое будет отображать фамилию и инициалы покупателей, скрывая другие поля.

```
mysql> CREATE OR REPLACE VIEW list_user
-> AS SELECT CONCAT(u_surname, " ",
-> SUBSTRING(u_name,1,1), ". ",
-> SUBSTRING(u_patronymic,1,1), ".") AS name
-> FROM users ORDER BY name;
Query OK, 0 rows affected (0.08 sec)
```

```
mysql> SELECT * FROM list_user;
```

```
+-----+
| name          |
+-----+
| Кузнецов М.П. |
| Корнеев А.А.  |
| Петров А.Ю.   |
| Иванов А.В.   |
| Лосев С.И.    |
| Симонов И.Н.  |
+-----+
```

```
6 rows in set (0.02 sec)
```

- 2. Создадим представление *price* с общей стоимостью книг в каждом каталоге.

```
mysql> CREATE VIEW price
-> AS SELECT b_catID, SUM(b_price*b_count) AS price
-> FROM books
-> GROUP BY b_catID
-> ORDER BY price;
Query OK, 0 rows affected (0.08 sec)
```

```
mysql> SELECT * FROM price;
```

b_catID	price
3	2908.00
2	4389.00
4	9514.00
1	12123.00
5	15177.00

• 5 rows in set (0.01 sec)

- Сформируем запрос к таблице *catalogs* и представлению *price*.

```
mysql> SELECT catalogs.cat_name, price.price
-> FROM price, catalogs
-> WHERE price.b_catID = catalogs.catID
-> ORDER BY catalogs.catID;
```

cat_name	price
Программирование	12123.00
Интернет	4389.00
Базы данных	2908.00
Сети	9514.00

- Сформируем запрос к представлению *price* (получение минимального и максимального значений стоимости книг в каталогах и общей стоимости всех книг).

```
mysql> SELECT MIN(price), MAX(price), SUM(price) FROM price;
```

MIN(price)	MAX(price)	SUM(price)
2908.00	15177.00	44111.00

• 1 row in set (0.02 sec)

- **Лабораторная работа № 10**

### Управление правами пользователей

- **Теоретические сведения**
- СУБД MySQL является многопользовательской средой, поэтому для доступа к таблицам БД могут быть созданы различные учетные записи с разным уровнем привилегий. Учетной записи редактора можно предоставить привилегии на просмотр таблицы, добавление новых записей и обновление уже существующих. Администратору БД можно предоставить более широкие полномочия (возможность создания таблиц, редактирования и удаления уже существующих). Для пользователя БД достаточно лишь просмотра таблиц.
- Рассмотрим следующие вопросы:
  - создание, редактирование и удаление учетных записей пользователей;
  - назначение и отмена привилегий.
    - Учетная запись является составной и принимает форму

'username' @ 'host', где *username* – имя пользователя, а *host* – наименование хоста, с которого пользователь может обращаться к серверу. Например, записи 'root' @ '127.0.0.1' и 'wet' @ '62.78.56.34' означают, что пользователь с именем *root* может обращаться с хоста, на котором расположен сервер, а *wet* – только с хоста с IP-адресом 62.78.56.34.

- IP-адрес 127.0.0.1 всегда относится к локальному хосту. Если сервер и клиент установлены на одном хосте, то сервер слушает соединения по этому адресу, а клиент отправляет на него SQL-запросы.

- IP-адрес 127.0.0.1 имеет псевдоним *localhost*, поэтому учетные записи вида 'root' @ '127.0.0.1' можно записывать в виде 'root' @ 'localhost'.

- Число адресов, с которых необходимо обеспечить доступ пользователю, может быть значительным. Для задания диапазона в имени хоста используется специальный символ "%". Так, учетная запись 'wet' @ '%' позволяет пользователю *wet* обращаться к серверу MySQL с любых компьютеров сети.

- Все учетные записи хранятся в таблице *user* системной базы данных с именем *mysql*. После первой инсталляции содержимое таблицы *user* выглядит так, как показано в листинге.

```
mysql> SELECT Host,User,Password FROM mysql.user;
```

Host	User	Password
localhost	root	
production.mysql.com	root	
127.0.0.1	root	
localhost		
production.mysql.com		

5 rows in set (0.27 sec)

- **Создание новой учетной записи.** Создать учетную запись позволяет оператор

- `CREATE USER 'username' @ 'host' [IDENTIFIED BY [PASSWORD] 'пароль'];`

- Оператор создает новую учетную запись с необязательным паролем. Если пароль не указан, в его качестве выступает пустая строка. Разумно хранить пароль в виде хэш-кода, полученного в результате необратимого шифрования. Чтобы воспользоваться этим механизмом авторизации, необходимо поместить между ключевым словом *IDENTIFIED BY* и паролем ключевое слово *PASSWORD*.

- **Удаление учетной записи.** Удалить учетную запись позволяет оператор

- `DROP USER 'username' @ 'host';`

- **Изменение имени пользователя в учетной записи.** Осуществляется с помощью оператора

- *RENAME USER* старое\_имя TO новое\_имя;

- **Назначение привилегий.** Рассмотренные выше операторы позволяют создавать, удалять и редактировать учетные записи, но они не позволяют изменять привилегии пользователя – сообщать MySQL, какой пользователь имеет право только на чтение информации, какой на чтение и редактирование, а кому предоставлены права изменять структуру БД и создавать учетные записи.

- Для решения этих задач предназначены операторы *GRANT* (назначает привилегии) и *REVOKE* (удаляет привилегии). Если учетной записи, которая показана в операторе *GRANT*, не существует, то она автоматически создается. Удаление всех привилегий с помощью оператора *REVOKE* не приводит к автоматическому уничтожению учетной записи. Для удаления пользователя необходимо воспользоваться оператором *DROP USER*.

- В простейшем случае оператор *GRANT* выглядит следующим образом:

```
mysql> GRANT ALL ON *.* TO 'wet'@'localhost' IDENTIFIED BY 'pass';  
Query OK, 0 rows affected (0.17 sec)
```

- Данный запрос создает пользователя с именем *wet* и паролем *pass*, который может обращаться к серверу с локального хоста (*localhost*) и имеет все права (*ALL*) для всех баз данных (*\*.\**). Если такой пользователь существует, то его привилегии будут изменены на *ALL*.

- Вместо ключевого слова *ALL* можно использовать любое из ключевых слов, представленных в табл. 9. Ключевое слово *ON* в операторе *GRANT* задает уровень привилегий, которые могут быть заданы на одном из четырех уровней, представленных в табл. 10. Для таблиц можно установить только следующие типы привилегий: *SELECT*, *INSERT*, *UPDATE*, *DELETE*, *CREATE*, *DROP*, *GRANT OPTION*, *INDEX* и *ALTER*. Это следует учитывать при использовании конструкции *GRANT ALL*, которая назначает привилегии на текущем уровне. Так, запрос уровня базы данных *GRANT ALL ON db.\** не предоставляет никаких глобальных привилегий.

- **Отмена привилегий.** Для отмены привилегий используется оператор *REVOKE*:

```
mysql> REVOKE DELETE, UPDATE ON *.* FROM 'wet'@'localhost';  
Query OK, 0 rows affected (0.02 sec)
```

- Оператор *REVOKE* отменяет привилегии, но не удаляет учетные записи, для их удаления необходимо воспользоваться оператором *DROP USER*.

Привилегия	Операция, разрешенная привилегией
<i>ALL [PRIVILEGES]</i>	Комбинация всех привилегий, за исключением привилегии <i>GRANT OPTION</i> , которая задается отдельно
<i>ALTER</i>	Позволяет редактировать таблицу с помощью оператора <i>ALTER TABLE</i>
<i>ALTER ROUTINE</i>	Позволяет редактировать или удалять хранимую процедуру
<i>CREATE</i>	Позволяет создавать таблицу при помощи оператора <i>CREATE TABLE</i>
<i>CREATE ROUTINE</i>	Позволяет создавать хранимую процедуру
<i>CREATE TEMPORARY TABLES</i>	Позволяет создавать временные таблицы
<i>CREATE USER</i>	Позволяет работать с учетными записями с помощью <i>CREATE USER</i> , <i>DROP USER</i> , <i>RENAME USER</i> и <i>REVOKE ALL PRIVILEGES</i>
<i>CREATE VIEW</i>	Позволяет создавать представление с помощью оператора <i>CREATE VIEW</i>
<i>DELETE</i>	Позволяет удалять записи при помощи оператора <i>DELETE</i>
<i>DROP</i>	Позволяет удалять таблицы при помощи оператора <i>DROP TABLE</i>
<i>EXECUTE</i>	Позволяет выполнять хранимые процедуры
<i>INDEX</i>	Позволяет работать с индексами, в частности, использовать операторы <i>CREATE INDEX</i> и <i>DROP INDEX</i>
<i>INSERT</i>	Позволяет добавлять в таблицу новые записи оператором <i>INSERT</i>
<i>LOCK TABLES</i>	Позволяет осуществлять блокировки таблиц при помощи операторов <i>LOCK TABLES</i> и <i>UNLOCK TABLES</i> . Для вступления в действие этой привилегии должна быть установлена привилегия <i>SELECT</i>
<i>SELECT</i>	Позволяет осуществлять выборки таблиц оператором <i>SELECT</i>
<i>SHOW DATABASES</i>	Позволяет просматривать список всех таблиц на сервере при помощи оператора <i>SHOW DATABASES</i>
<i>SHOW VIEW</i>	Позволяет использовать оператор <i>SHOW CREATE VIEW</i>
<i>UPDATE</i>	Позволяет обновлять содержимое таблиц оператором <i>UPDATE</i>
<i>USAGE</i>	Синоним для статуса «отсутствуют привилегии»
<i>GRANT OPTION</i>	Позволяет управлять привилегиями других пользователей, без данной привилегии невозможно выполнить операторы <i>GRANT</i> и <i>REVOKE</i>

• Таблица 10

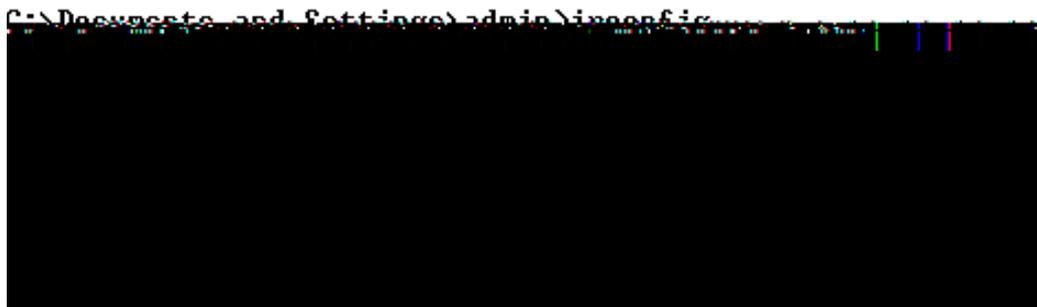
Ключевое слово ON	Уровень
<i>ON *.*</i>	Глобальный уровень – пользователь с полномочиями на глобальном уровне может обращаться ко всем БД и таблицам, входящим в их состав
<i>ON db.*</i>	Уровень базы данных – привилегии распространяются на таблицы базы данных <i>db</i>
<i>ON db.tbl</i>	Уровень таблицы – привилегии распространяются на таблицу <i>tbl</i> базы данных <i>db</i>
<i>ON db.tbl</i>	Уровень столбца – привилегии касаются отдельных столбцов в таблице <i>tbl</i> базы данных <i>db</i> . Список столбцов указывается в скобках через запятую после ключевых слов <i>SELECT</i> , <i>INSERT</i> , <i>UPDATE</i>

• Практическая работа

- При выполнении лабораторной работы необходимо:
  - создать учетную запись нового пользователя и наделить его определенными привилегиями;
  - составить отчет по лабораторной работе.

### • Пример выполнения работы

- Для работы выберем два компьютера, подключенных к локальной сети. На одном необходимо развернуть сервер MySQL, на другой – скопировать клиент командной строки *mysql.exe*. Определим IP-адрес сервера:



- Создадим новую учетную запись, позволив пользователю *user1* обращаться к серверу MySQL с любых компьютеров сети:

```
mysql> CREATE USER 'user1'@'%' IDENTIFIED BY '123';
Query OK, 0 rows affected (0.01 sec)
```

- Назначим этому пользователю привилегии глобального уровня:

```
mysql> GRANT ALL ON *.* TO 'user1'@'%' IDENTIFIED BY '123';
Query OK, 0 rows affected (0.00 sec)
```

- На клиентском компьютере в командной строке (например, с помощью FAR), запустим клиент командной строки в следующем формате:

```
The FAR manager, version 1.70 beta 5 (build 1634)
Copyright (C) 1996-2000 Eugene Roshal, Copyright (C) 2000-2003 FAR Gro
Зарегистрирован: xUSSR регистрация
C:\>mysql -u user1 -p123 -h 192.168.67.6
```

- Наблюдаем отклик удаленного сервера и работаем с ним как обычно:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.0.51b-community-nt MySQL Community Edition (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| book |
| mysql |
| test |
+-----+
4 rows in set (0.00 sec)
```

### • 4. ВАРИАНТЫ ЗАДАНИЙ К ЛАБОРАТОРНЫМ РАБОТАМ

- 1. *Страховая компания.* Страховая компания имеет филиалы, которые характеризуются наименованием, адресом и телефоном. В филиалы обращаются клиенты с целью заключения договора о страховании. В зависимости от принимаемых на страхование объектов и страхуемых рисков договор заключается по определенному виду страхования (страхование автотранспорта от угона, страхование домашнего имущества, добровольное медицинское страхование). При заключении договора фиксируются: дата заключения, страховая сумма, вид страхования, тарифная ставка и филиал, в котором заключался договор. Договоры заключают страховые агенты. Помимо информации об агентах (фамилия, имя, отчество, адрес, телефон) нужно хранить филиал, в котором они работают. Необходимо иметь возможность рассчитывать заработную плату агентам. Заработная плата составляет некоторый процент от страхового платежа (платеж – страховая сумма, умноженная на тарифную ставку). Процент зависит от вида страхования, по которому заключен договор.

- 2. *Гостиница.* Гостиница предоставляет номера клиентам. Каждый номер характеризуется вместимостью, комфортностью (люкс, полуплюкс, обычный) и ценой. О клиентах собирается определенная информация (фамилия, имя, отчество, паспортные данные, адрес жительства и некоторый комментарий). Сдача номера клиенту производится при наличии свободных мест в номерах, подходящих клиенту по указанным выше параметрам. При заселении фиксируется дата заселения. При выезде из гостиницы для каждого места запоминается дата освобождения. Необходимо также осуществлять бронирование номеров. Для постоянных клиентов, а также для определенных категорий клиентов предусмотрена система скидок. Скидки могут суммироваться.

- 3. *Ломбард.* В ломбард обращаются различные лица с целью получения денежных средств под залог товаров. Клиент сообщает фамилию, имя, отчество и другие паспортные данные. После оценивания стоимости принесенного в качестве залога товара работник ломбарда определяет сумму, которую готов выдать на руки клиенту, а также свои комиссионные. Кроме того определяется срок возврата денег. Если клиент согласен, то договоренности фиксируются в виде документа, деньги выдаются клиенту, а товар остается в ломбарде. Если в указанный срок не происходит возврата денег, товар переходит в собственность ломбарда. После перехода прав собственности на товар, ломбард может продавать товары по цене, меньшей или большей, чем была заявлена при сдаче. Цена может меняться несколько раз, в зависимости от ситуации на рынке (например, владелец ломбарда может устроить распродажу зимних вещей в конце зимы). Помимо текущей цены нужно хранить все

возможные значения цены для данного товара.

- 4. *Оптово-розничная продажа товаров.* Компания торгует товарами из определенного спектра. Каждый товар характеризуется наименованием, оптовой ценой, розничной ценой и справочной информацией. В компанию обращаются покупатели, для каждого из которых в базе данных фиксируются стандартные данные (наименование, адрес, телефон, контактное лицо). По каждой сделке составляется документ, в котором наряду с покупателем фиксируются количество купленного им товара и дата покупки. Обычно покупатели в рамках одной сделки покупают не один товар, а сразу несколько. Также компания решила предоставлять скидки в зависимости от количества закупленных товаров и их общей стоимости.

- 5. *Ведение заказов.* Компания занимается оптовой продажей различных товаров. Каждый из товаров характеризуется ценой, справочной информацией и признаком наличия или отсутствия доставки. В компанию обращаются заказчики. Для каждого из них в базе данных запоминаются стандартные данные (наименование, адрес, телефон, контактное лицо). По каждому заказу составляется документ, в котором наряду с заказчиком фиксируются количество купленного им товара и дата покупки. Доставка товаров может производиться способами, различными по цене и скорости. Нужно хранить информацию о том, какими способами может осуществляться. Все виды дополнительных работ и определена сумма оплаты по факту их выполнения. При возникновении дополнительной работы назначается ответственный, фиксируется дата начала. По факту окончания фиксируется дата и выплачивается дополнительная сумма к зарплате с учетом классификации. Необходимо учесть разделение сотрудников на преподавателей и учебно-вспомогательный персонал. Для первых нужно хранить сведения об ученой степени и ученом звании, для вторых – о должности. Некоторые работы являются трудоемкими, иные данные) также фиксируются в базе данных. По каждому факту удовлетворения интересов обеих сторон составляется документ. В документе указываются соискатель, работодатель, должность и комиссионные (доход бюро). В базе должна фиксироваться не только сделка, но и информация по открытым вакансиям. Кроме того для автоматического поиска вариантов необходимо вести справочник «Виды деятельности».

- 7. *Нотариальная контора.* Нотариальная контора готова предоставить клиенту определенный комплекс услуг. Услуги формализованы, т. е. составлен их список с описанием каждой услуги. При обращении клиента его стандартные данные (название, вид деятельности, адрес, телефон) фиксируются в базе данных. По каждому факту оказания услуги клиенту составляется документ, в

котором указываются дата, услуга, сумма сделки, комиссионные (доход конторы), описание сделки. В рамках одной сделки клиенту может быть оказано несколько услуг. Стоимость каждой услуги фиксирована. Кроме того компания предоставляет в рамках одной сделки различные виды скидок. Скидки могут суммироваться.

- 8. *Фирма по продаже запчастей.* Фирма продает запасные части для автомобилей. Фирма имеет определенный набор поставщиков, по которым известны название, адрес и телефон. У поставщиков приобретаются детали. Каждая деталь характеризуется названием, артикулом и ценой. Некоторые из поставщиков могут поставлять одинаковые детали (один артикул). Каждый факт покупки запчастей у поставщика фиксируется в базе данных, причем обязательными для запоминания являются дата покупки и количество приобретенных деталей. Цена детали может меняться от поставки к поставке. Поставщики заранее ставят фирму в известность о дате изменения цены и ее новом значении. Нужно хранить не только текущее значение цены, но и всю историю изменения цен.

- 9. *Курсы по повышению квалификации.* В учебном заведении организованы курсы повышения квалификации. Группы слушателей формируются в зависимости от специальности и отделения. В каждую из них включено определенное количество слушателей. Проведение занятий обеспечивает штат преподавателей, для каждого из которых в базе данных зарегистрированы стандартные анкетные данные (фамилия, имя, отчество, телефон) и стаж работы. В результате распределения нагрузки получена информация о том, сколько часов занятий проводит каждый преподаватель с соответствующими группами. Хранятся также сведения о виде занятий (лекция, практика), дисциплине и оплате за 1 час. Размер почасовой оплаты зависит от предмета и типа занятия. Кроме того каждый преподаватель может вести не все предметы, а только некоторые.

- 10. *Определение факультативов для студентов.* Преподаватели кафедры в высшем учебном заведении обеспечивают проведение факультативных занятий по некоторым предметам. Имеются сведения о студентах, включающие стандартные анкетные данные (фамилия, имя, отчество, группа, адрес, телефон). По каждому факультативу существует определенное количество часов и вид проводимых занятий (лекции, практика, лабораторные работы). В результате работы со студентами появляется информация о том, кто из них записался на какие факультативы. Существует некоторый минимальный объем факультативных предметов, которые должен прослушать каждый студент. По окончании семестра в базу данных заносится информация об оценках, полученных студентами на экзаменах. Некоторые из факультативов могут длиться более одного семестра. В каждом семестре для предмета устанавливается объем

лекций, практик и лабораторных работ в часах. В качестве итоговой оценки за предмет берется последняя оценка, полученная студентом.

- *11. Распределение учебной нагрузки.* Необходимо распределять нагрузку между преподавателями кафедры. Имеются сведения о преподавателях, включающие наряду с анкетными данными сведения об их ученой степени, занимаемой должности и стаже работы. Преподаватели кафедры должны обеспечить проведение занятий по некоторым дисциплинам. По каждой из них существует определенное количество часов. В результате распределения нагрузки необходимо получить информацию следующего рода: «Такой-то преподаватель проводит занятия по такой-то дисциплине с такой-то группой». Все проводимые занятия делятся на лекционные и практические. По каждому виду занятий устанавливается свое количество часов. Кроме того данные по нагрузке нужно хранить несколько лет.

- *12. Распределение дополнительных обязанностей.* Кафедра вуза имеет штат сотрудников, каждый из которых получает определенный оклад. Каждый месяц возникает потребность в выполнении некоторой дополнительной работы, не входящей в круг основных обязанностей сотрудников. Для наведения порядка в этой сфере классифицированы все виды дополнительных работ и определена сумма оплаты по факту их выполнения. При возникновении дополнительной работы назначается ответственный, фиксируется дата начала. По факту окончания фиксируется дата и выплачивается дополнительная сумма к зарплате с учетом классификации. Необходимо учесть разделение сотрудников на преподавателей и учебно-вспомогательный персонал. Для первых нужно хранить сведения об ученой степени и ученом звании, для вторых – о должности. Некоторые работы являются трудоемкими и срочными, что требует привлечения к их выполнению нескольких сотрудников. Длительность работ различна. Нужно заранее планировать длительность работы и количество сотрудников, занятых для выполнения работы.

- *13. Техническое обслуживание станков.* Компания занимается ремонтом станков и другого оборудования. Клиентами компании являются промышленные предприятия. Ремонтные работы организованы следующим образом: все станки классифицированы по типам, странам-производителям, годам выпуска и маркам. Все виды ремонта отличаются названием, продолжительностью в днях, стоимостью. Исходя из этих данных, по каждому факту ремонта фиксируется вид станка, дата начала и дата окончания ремонта. Анализ показал, что нужно не просто подразделять станки по типам, а иметь информацию о том, сколько раз ремонтировался тот или иной станок.

- *14. Туристическая фирма.* Фирма продает путевки клиентам. У

каждого клиента запрашиваются стандартные данные – фамилия, имя, отчество, адрес, телефон. После этого сотрудники компании выясняют у клиента стаж работы, некоторые из них удостоены различных званий. В рамках одного спектакля на одну и ту же роль привлекается несколько актеров. Контракт определяет базовую зарплату актера, а по итогам реально отыгранных спектаклей актеру назначается премия. В базе данных нужно хранить информацию за несколько лет.

- 23. Платная поликлиника. В поликлинике работают врачи различных специальностей, имеющие разную квалификацию. Каждый день в поликлинику обращаются пациенты. Все пациенты проходят обязательную регистрацию в странах. Путевки продаются на одну, две или четыре недели. Стоимость путевки зависит от длительности тура и отеля.

- 15. *Грузовые перевозки.* Компания осуществляет перевозки грузов по различным маршрутам. Необходимо отслеживать стоимость перевозок с учетом заработной платы водителей. Для каждого маршрута определено название, вычислено примерное расстояние и установлена некоторая оплата для водителя. Информация о водителях включает фамилию, имя, отчество и стаж. Для проведения расчетов хранится полная информация о перевозках (маршрут, водитель, даты отправки и прибытия). По факту некоторых перевозок водителям выплачивается премия. Фирма решила ввести гибкую систему оплаты. Оплата водителям должна зависеть не только от маршрута, но и от стажа водителя. Кроме того, нужно учесть, что перевозку могут осуществлять два водителя.

- 16. *Учет телефонных переговоров.* Телефонная компания предоставляет абонентам телефонные линии для междугородних переговоров. Абонентами компании являются юридические лица, имеющие телефонную точку, ИНН, расчетный счет в банке. Стоимость переговоров зависит от города, в который осуществляется звонок, и времени суток (день, ночь). Каждый звонок абонента автоматически фиксируется в базе данных. При этом запоминаются город, дата, длительность разговора и время суток. Компания решила ввести гибкую систему скидок. Так, стоимость минуты теперь уменьшается в зависимости от длительности разговора. Размер скидки для каждого города разный.

- 17. *Учет внутриофисных расходов.* Сотрудники частной фирмы могут осуществлять мелкие покупки для нужд фирмы, предоставляя в бухгалтерию товарный чек. Бухгалтерия отслеживает внутриофисные расходы. Фирма состоит из отделов, каждый из которых имеет название. В каждом отделе работает определенное количество сотрудников. Сотрудники могут осуществлять покупки в соответствии с видами расходов. Каждый вид расходов имеет

название, некоторое описание и предельную сумму средств, которые могут быть потрачены по данному виду расходов в месяц. При каждой покупке сотрудник оформляет документ, где указывает вид расхода, дату, сумму и отдел. Нужно хранить данные о расходах не только в целом по отделу, но и по отдельным сотрудникам. Нормативы по расходованию средств устанавливаются не в целом, а по каждому отделу за каждый месяц. Неиспользованные в текущем месяце деньги могут быть использованы позже.

- *18. Библиотека.* Библиотека решила зарабатывать деньги, выдавая напрокат книги, имеющиеся в небольшом количестве экземпляров. У каждой книги, выдаваемой в прокат, есть название, автор, жанр. В зависимости от ценности книги для каждой из них определена залоговая стоимость (сумма, вносимая клиентом при взятии книги напрокат) и стоимость проката (сумма, которую клиент платит при возврате книги, получая назад залог). Читатели регистрируются в картотеке, которая содержит стандартные анкетные данные (фамилия, имя, отчество, адрес, телефон). Каждый читатель может обращаться в библиотеку несколько раз. Все обращения читателей фиксируются, при этом по каждому факту выдачи книги запоминаются дата выдачи и ожидаемая дата возврата. Стоимость проката книги должна зависеть не только от самой книги, но и от срока ее проката. Кроме того, необходимо добавить систему штрафов за вред, нанесенный книге, и систему скидок для некоторых категорий читателей.

- *19. Прокат автомобилей.* Фирма, занимающаяся прокатом автомобилей, имеет автопарк, содержащий некоторое количество автомобилей различных марок, стоимостей и типов. Каждый автомобиль имеет свою стоимость проката. В пункт проката обращаются клиенты. Клиенты проходят обязательную регистрацию, в ходе которой о них собирается стандартная информация (фамилия, имя, отчество, адрес, телефон). Каждый клиент может обращаться в пункт проката несколько раз. Обращения клиентов фиксируются, при этом по каждой сделке запоминаются дата выдачи и ожидаемая дата возврата. Стоимость проката автомобиля должна зависеть не только от самого автомобиля, но и от срока его проката, а также от года выпуска. Также нужно ввести систему штрафов за возвращение автомобиля в ненадлежащем виде и систему скидок для постоянных клиентов.

- *20. Выдача банком кредитов.* Коммерческий банк выдает кредиты юридическим лицам. В зависимости от условий получения кредита, процентной ставки и срока возврата все кредитные операции делятся на несколько основных видов. Каждый из этих видов имеет свое название. Кредит может получить юридическое лицо (клиент), при регистрации предоставивший следующие сведения: название, вид

собственности, адрес, телефон, контактное лицо. Каждый факт выдачи кредита регистрируется банком, при этом фиксируются сумма кредита, клиент и дата выдачи. Чтобы отслеживать динамику возврата кредитов принято решение учитывать в системе еще и дату фактического возврата денег. Нужно еще учесть, что кредит может гаситься частями, и за задержку возврата кредита начисляются штрафы.

- 21. *Инвестиционная компания.* Компания занимается вложением денежных средств в ценные бумаги, которые характеризуются рейтингом, доходностью за прошлый год, минимальной суммой сделки и некоторой дополнительной информацией. Клиентами компании являются предприятия, которые доверяют ей управлять их свободными денежными средствами на определенный период. Необходимо выбрать вид ценных бумаг, которые позволят получить прибыль и компании и клиенту. При работе с клиентом существенной является информация о предприятии – название, вид собственности, адрес и телефон. Каждая инвестиция характеризуется информацией о клиенте, информацией о ценной бумаге, котировкой бумаги, датой ее покупки и датой ее продажи. Необходимо хранить историю котировок каждой ценной бумаги. Кроме того, помимо вложений в ценные бумаги существует возможность вкладывать деньги в банковские депозиты.

- 22. *Занятость актеров театра.* Коммерческий директор театра организует привлечение актеров и заключение контрактов. Каждый год театр осуществляет постановку различных спектаклей. Каждый спектакль имеет определенный бюджет. Для участия в конкретных постановках в определенных ролях привлекаются актеры. С каждым из актеров заключается персональный контракт на определенную сумму. Каждый актер имеет стаж работы, некоторые из них удостоены различных званий. В рамках одного спектакля на одну и ту же роль привлекается несколько актеров. Контракт определяет базовую зарплату актера, а по итогам реально отыгранных спектаклей актеру назначается премия. В базе данных нужно хранить информацию за несколько лет.

- 23. *Платная поликлиника.* В поликлинике работают врачи различных специальностей, имеющие разную квалификацию. Каждый день в поликлинику обращаются пациенты. Все пациенты проходят обязательную регистрацию, при которой в базу данных заносятся стандартные анкетные данные (фамилия, имя, отчество, год рождения, адрес). При обращении в поликлинику пациент обследуется и проходит лечение у разных специалистов. Каждый пациент может обращаться в поликлинику несколько раз, нуждаясь в различной медицинской помощи. Все обращения пациентов фиксируются, при этом устанавливается диагноз, определяется

стоимость лечения, запоминается дата обращения. Общая стоимость лечения зависит от стоимости консультаций и процедур, назначенных пациенту. Для определенных категорий граждан предусмотрены скидки.

- 24. *Анализ динамики показателей финансовой отчетности предприятий.* Информационно-аналитический центр крупного холдинга отслеживает динамику показателей предприятий холдинга. В структуру холдинга входят несколько предприятий. Каждое предприятие имеет стандартные характеристики (название, реквизиты, телефон, контактное лицо). Работа предприятия может быть оценена следующим образом: в начале каждого отчетного периода на основе финансовой отчетности вычисляется определенный набор показателей. Важность показателей характеризуется некоторыми числовыми константами. Значение каждого показателя измеряется в некоторой системе единиц. Некоторые показатели считаются в рублях, некоторые в долларах, некоторые в евро. Для удобства работы с показателями нужно хранить изменения курсов валют относительно друг друга.

- 25. *Учет телекомпанией стоимости прошедшей в эфире рекламы.* Работа коммерческой службы телевизионной компании построена следующим образом: заказчики просят поместить свою рекламу в определенной передаче в определенный день. Каждый рекламный ролик имеет определенную продолжительность. Для каждой организации-заказчика известны банковские реквизиты, телефон и контактное лицо для проведения переговоров. Передачи имеют определенный рейтинг. Стоимость минуты рекламы в каждой передаче определяется, исходя из рейтинга передачи и прочих соображений. Необходимо хранить информацию об агентах, заключивших договоры на рекламу. Зарплата рекламных агентов составляет некоторый процент от общей стоимости рекламы, прошедшей в эфире.

- 26. *IT-компания.* Компания оказывает IT-услуги организациям и предприятиям. В компании работают сотрудники, о которых должна сохраняться стандартная информация и данные о квалификации (владение языками и системами программирования, знание СУБД, операционных систем). В компанию обращаются клиенты, о которых собираются стандартные данные (наименование и адрес организации, телефон, адрес электронной почты, фамилия, имя и отчество контактного лица для связи). Задание для клиента выполняет определенный сотрудник, при этом фиксируется дата выдачи задания и трудоемкость выполнения (в часах). При повторном обращении клиент переходит в категорию постоянных и получает скидку. С ростом компании возникла необходимость разделения ее на отделы. Увеличились масштабы проектов, и теперь задание клиента

поручается отделу. В рамках одного договора может выполняться несколько заданий разными отделами компании.

- 27. *Ювелирная мастерская.* Ювелирная мастерская осуществляет изготовление ювелирных изделий для частных лиц на заказ. Мастерская работает с определенными материалами (платина, золото, серебро, драгоценные камни). При обращении потенциального клиента выясняется, какое именно изделие ему необходимо. Все изготавливаемые изделия принадлежат к некоторому типу (серьги, кольца, броши, браслеты), выполняются из определенного материала, имеют некоторый вес и цену (включающую стоимость материалов и работы). Ювелирное изделие может состоять из нескольких материалов. Кроме того, постоянным клиентам мастерская предоставляет скидки.

- 28. *Парикмахерская.* Парикмахерская стрижет клиентов в соответствии с их пожеланиями и некоторым каталогом различных видов стрижки. Для каждой стрижки определены название, категория (мужская, женская, детская), стоимость работы. Для наведения порядка составляется база данных клиентов, где запоминаются их анкетные данные (фамилия, имя, отчество). Начиная с 5-ой стрижки, клиент переходит в категорию постоянных и получает скидку в 3 % при каждой последующей стрижке. После того, как закончена очередная работа, в БД фиксируются стрижка, клиент и дата производства работ. Кроме того, у парикмахерской появился филиал и необходима отдельная статистика по филиалам. Стоимость стрижки может меняться с течением времени. Нужно хранить не только последнюю цену, но и все данные по изменению цены стрижки.

- 29. *Химчистка.* Химчистка осуществляет прием у населения вещей для выведения пятен. Для наведения порядка составляется база данных клиентов, в которой запоминаются их анкетные данные (фамилия, имя, отчество, адрес, телефон). Начиная с 3-го обращения, клиент переходит в категорию постоянных клиентов и получает скидку в 3 % при чистке каждой последующей вещи. Все оказываемые услуги подразделяются на виды, имеющие название, тип и стоимость, зависящую от сложности работ. Работа с клиентом первоначально состоит в определении объема работ, вида услуги и, соответственно, ее стоимости. Если клиент согласен, он оставляет вещь (при этом фиксируется услуга, клиент и дата приема) и забирает ее после обработки (при этом фиксируется дата возврата). Химчистка заключает с клиентом договор. Клиент может одновременно сдавать в чистку несколько вещей. У химчистки появились филиалы, и необходима отдельная статистика по филиалам. Введены надбавки за срочность и сложность.

- 30. *Сдача в аренду торговых площадей.* Торговый центр сдает в аренду коммерсантам свои торговые площади. В результате

планирования определено некоторое количество торговых точек в пределах здания, которые могут сдаваться в аренду. Для каждой из торговых точек важными данными являются этаж, площадь, наличие кондиционера и стоимость аренды в день. С потенциальных клиентов собираются стандартные данные (название, адрес, телефон, реквизиты, контактное лицо). При появлении потенциального клиента ему показывают имеющиеся свободные площади. При достижении соглашения оформляется договор и в базе данных фиксируется торговая точка, клиент, период (срок) аренды. Некоторые клиенты в рамках одного договора арендуют сразу несколько торговых точек, причем для каждой точки возможен свой срок аренды. Дата заключения договора может не совпадать с датой начала аренды. Необходимо собирать информацию об ежемесячных платежах, поступающих от арендаторов.

- 
- 

- 
- 
-

## • БИБЛИОГРАФИЧЕСКИЙ СПИСОК

- 
- **Основные источники:**
- 1. Кумскова, И.А. Базы данных : учебник / Кумскова И.А. — Москва : КноРус, 2020. — 400 с. — (СПО). — ISBN 978-5-406-07467-1. — URL: <https://book.ru/book/932493> (дата обращения: 21.10.2020). — Текст : электронный.
- 2. Кондрашов, Ю.Н. Язык SQL. Сборник ситуационных задач по дисциплине «Базы данных : учебно-практическое пособие / Кондрашов Ю.Н. — Москва : Русайнс, 2020. — 125 с. — ISBN 978-5-4365-4598-1. — URL: <https://book.ru/book/935744> (дата обращения: 21.10.2020). — Текст : электронный.
- 3. Астахова, И.Ф. Объектные базы данных : учебное пособие / Астахова И.Ф., Борисенков Д.В., Киселева Е.И., Самойлов Н.К. — Москва : Русайнс, 2020. — 93 с. — ISBN 978-5-4365-5404-4. — URL: <https://book.ru/book/936907> (дата обращения: 21.10.2020). — Текст : электронный.
- 4. Чулюков, В.А. Проектирование баз данных. Практический курс : учебное пособие / Чулюков В.А., Астахова И.Ф., Башарина С.О., Сидорова О.А. — Москва : Русайнс, 2020. — 163 с. — ISBN 978-5-4365-5748-9. — URL: <https://book.ru/book/938011> (дата обращения: 21.10.2020). — Текст : электронный.
- 5. Сидорова, Н. П. Базы данных: практикум по проектированию реляционных баз данных : учебное пособие / Н. П. Сидорова. — Королёв : МГОТУ, 2020. — 92 с. — ISBN 978-5-4499-0799-8. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/149436> (дата обращения: 22.10.2020). — Режим доступа: для авториз. пользователей.
- 6. Каминский, В. Н. Базы данных : учебное пособие / В. Н. Каминский. — Санкт-Петербург : БГТУ "Военмех" им. Д.Ф. Устинова, 2017. — 106 с. — ISBN 978-5-906920-36-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/121826> (дата обращения: 22.10.2020). — Режим доступа: для авториз. пользователей.
- **Интернет ресурсы:**
- 1. Электронный ресурс. URL: <http://www.intuit.ru>
- 2. Электронный ресурс. URL: <http://www.edu.bpwin.ru>

- ПРИЛОЖЕНИЯ

- Приложение 1. Основные математические функции MySQL

Обозначение	Описание
<i>ABS(X)</i>	Возвращает абсолютное значение аргумента <i>X</i>
<p><i>ASCII(str)</i> Описание</p> <p><i>ACOS(X)</i> Возвращает арккосинус аргумента <i>X</i> или <i>NULL</i>, если значение <i>X</i> не находится в диапазоне от -1 до 1</p> <p><i>ASIN(X)</i> Возвращает арксинус аргумента <i>X</i> или <i>NULL</i>, если значение <i>X</i> не находится в диапазоне от -1 до 1</p> <p><i>ATAN(X)</i> Возвращает арктангенс аргумента <i>X</i></p> <p><i>CEIL(X)</i> Принимает дробный аргумент <i>X</i> и возвращает первое целое число, находящееся справа от аргумента</p> <p><i>COS(X)</i> Вычисляет косинус угла <i>X</i>, заданного в радианах</p> <p><i>COT(X)</i> Вычисляет котангенс угла <i>X</i>, заданного в радианах</p> <p><i>DEGREES(X)</i> Возвращает значение угла <i>X</i>, преобразованное из радиан в градусы</p> <p><i>EXP(X)</i> Вычисляет значение <math>e^x</math></p> <p><i>FLOOR(X)</i> Принимает дробный аргумент <i>X</i> и возвращает первое целое число, находящееся слева от аргумента</p>	<p>Возвращает значение ASCII-кода первого символа строки <i>str</i>. Для пустой строки возвращается значение 0</p>

*LN(X)*

Вычисляет натуральный логарифм числа  $X$

*LOG2(X)*

Вычисляет логарифм числа  $X$  по основанию 2

*LOG10(X)*

Вычисляет десятичный логарифм числа  $X$

*MOD(M,N)*

Возвращает остаток от деления целого числа  $M$  на целое число  $N$

*PI( )*

Используется без аргументов. Возвращает число  $\pi$

*POW(X,Y)*

Возвращает значение числа  $X$ , возведенного в степень  $Y$

*RADIANS(X)*

Возвращает значение угла  $X$ , преобразованное из градусов в радианы

*RAND(X)*

Возвращает случайное значение с плавающей точкой в диапазоне от 0.0 до 1.0

*ROUND(X)*

Возвращает округленное до ближайшего целого значение числа  $X$

*SIGN(X)*

Позволяет определить знак числа  $X$ . Возвращает -1, 0 или 1, если  $X$  отрицательно, равно нулю или положительно

*SIN(X)*

Вычисляет синус угла  $X$ , заданного в радианах

*SQRT(X)*

Вычисляет квадратный

<p>корень числа <math>X</math></p> <p><math>TAN(X)</math> Вычисляет тангенс угла <math>X</math>, заданного в радианах</p> <p><math>TRUNCATE(X,D)</math> Возвращает число <math>X</math> с дробной частью, имеющей <math>D</math> знаков после запятой. Если количество знаков в <math>X</math> больше <math>D</math>, лишние разряды усекаются. Если меньше, то в конец числа добавляются нули</p> <p>Приложение 2. Основные строковые функции MySQL</p> <p><b>Обозначение</b></p>	
$BIN(N)$	Принимает десятичное число $N$ и возвращает его двоичное представление
$BIT\_LENGTH(str)$	Принимает строку $str$ и возвращает ее длину в битах
$CHAR(N1, N2, \dots)$	Принимает последовательность из ASCII-кодов и возвращает строку, построенную путем объединения соответствующих им символов
$CHAR\_LENGTH(str)$	Принимает строку $str$ и возвращает число символов в строке
$CHARSET(str)$	Возвращает имя кодировки, в которой представлена строка
$COLLATION(str)$	Возвращает порядок сортировки, установленный для кодировки аргумента $str$
$CONCAT(str1, str2, \dots)$	Возвращает строку, созданную путем объединения всех аргументов, количество которых не ограничено. Если хотя бы один аргумент равен $NULL$ , то возвращается значение $NULL$
$CONCAT\_WS(separator, str1, str2, \dots)$	Также объединяет аргументы $str1, str2$ и т.д., помещая между ними разделитель $separator$
$CONV(N, from\_base, to\_base)$	Преобразует число $N$ из одной системы счисления $from\_base$ в другую $to\_base$ . Параметры $from\_base$ и $to\_base$ могут принимать значения от 2 до 36
$ELT(N, str1, str2, \dots)$	Возвращает $N$ -ю строку из списка аргументов $str1, str2, \dots$ (для $N=1$ возвращается $str1$ , для $N=2$ – $str2$ и т.д.)
$FIELD(str, str1, str2, \dots)$	Находит строку $str$ в списке $str1, str2, \dots$ и возвращает номер строки в этом списке (нумерация начинается с 1)
<p>Возвращает <math>len</math> крайних левых символов строки <math>str</math></p> <p><math>LENGTH(str)</math> Возвращает длину строки <math>str</math></p> <p><math>LOCATE(substr, str [, pos])</math> Возвращает позицию первого вхождения подстроки <math>substr</math> в строку <math>str</math>.</p>	<p><math>INDEX(str, str\_list)</math> Ищет вхождение строки <math>str</math> в список <math>str\_list</math> и возвращает номер строки в <math>str\_list</math></p> <p><math>PERIOD\_FOR\_DAY(N)</math> Принимает число дней <math>N</math>, прошедших с нулевого года, и возвращает дату в формате 'YYYY-MM-DD'. Обычно используется совместно с функцией <math>TO\_DAYS(date)</math></p> <p><math>HOUR(datetime)</math> Извлекаем из значения <math>datetime</math> часы (от 0 до 23)</p>

При наличии необязательного аргумента <i>pos</i> поиск начинается с позиции, указанной в этом аргументе <i>FIND_IN_SET(str, str_list)</i>	<p><i>LAST_DAY(datetime)</i> Принимает дату и время суток и возвращает дату – последний день текущего месяца</p> <p><i>MAKEDATE(year, dayofyear)</i> Принимает год <i>year</i> и номер дня в году <i>dayofyear</i> и возвращает дату в формате 'YYYY-ММ-DD'</p> <p><i>MAKETIME(hour, minute, second)</i> Принимает час <i>hour</i>, минуты <i>minute</i> и секунды и возвращает позицию первого вхождения подстроки <i>substr</i> в строку <i>str</i></p>
<i>LOWER(str)</i> <i>LEFT(str, len)</i>	Возвращает строку <i>str</i> , записанную строчными символами
<i>LPAD(str, len, padstr)</i>	Возвращает строку <i>str</i> , дополненную слева строкой <i>padstr</i> до длины <i>len</i> символов. Если строка содержит более <i>len</i> символов, то она усекается до <i>len</i>
<i>LTRIM(str)</i>	Возвращает строку <i>str</i> , в которой удалены все начальные пробелы
<i>MID(str, pos [, len])</i>	Возвращает подстроку строки <i>str</i> , которая начинается с позиции <i>pos</i> и имеет длину <i>len</i> символов. Если параметр <i>len</i> не указывается, то подстрока возвращается, начиная с позиции <i>pos</i> и до конца строки <i>str</i>
<i>OCT(N)</i>	Принимает десятичное число <i>N</i> и возвращает его в восьмеричной системе счисления
<i>ORD(str)</i>	Возвращает значение ASCII-кода первого символа строки <i>str</i> . В отличие от функции <i>ASCII()</i> корректно работает с многобайтными кодировками
<i>REPEAT(str, count)</i>	Возвращает строку, полученную из <i>count</i> повторений строки <i>str</i>
<i>REPLACE(str, from_str, to_str)</i>	Возвращает строку <i>str</i> , в которой все подстроки <i>from_str</i> заменены <i>to_str</i>
<i>REVERSE(str)</i>	Возвращает строку <i>str</i> , записанную в обратном порядке
<i>RIGHT(str, len)</i>	Возвращает <i>len</i> крайних правых символов строки <i>str</i> , или всю строку, если аргумент <i>len</i> равен <i>NULL</i> или меньше 1
<i>RPAD(str, len, padstr)</i>	Возвращает строку <i>str</i> , дополненную справа строкой <i>padstr</i> до длины <i>len</i> символов
<i>RTRIM(str)</i>	Возвращает строку <i>str</i> , в которой удалены все конечные пробелы
<i>SPACE(N)</i>	Возвращает строку, состоящую из <i>N</i> пробелов, или пустую строку, если <i>N</i> имеет отрицательное значение
<i>SUBSTRING_INDEX(str, delim, N)</i>	Возвращает подстроку строки <i>str</i> . Если параметр <i>N</i> имеет положительное значение, то функция находит <i>N</i> -е вхождение (отсчет слева) подстроки <i>delim</i> в строку <i>str</i> и возвращает всю часть строки, расположенную слева от подстроки <i>delim</i> . Если <i>N</i> имеет отрицательное значение, то функция находит <i>N</i> -е вхождение (отсчет справа) подстроки <i>delim</i> в строку <i>str</i> и возвращает часть строки, расположенную справа от подстроки <i>delim</i>
<i>TRIM([[BOTH   LEADING   TRAILING ] [remstr] FROM ]</i>	Удаляет из строки <i>str</i> расположенные в начале (в конце) символы, указанные в строке <i>remstr</i> . Если указано ключевое

<i>str</i> )	слово <i>LEADING</i> , удаляются расположенные в начале символы, если <i>TRAILING</i> – в конце, если <i>BOTH</i> – и в начале и в конце. Если ключевые слова не заданы, по умолчанию принимается <i>BOTH</i> . Если строка <i>remstr</i> не задана, то в качестве удаляемых символов выступают пробелы
<i>UNHEX(str)</i>	Является обратной функции <i>HEX( )</i> и интерпретирует каждую пару символов строки <i>str</i> как шестнадцатеричный код, который необходимо преобразовать в символ
<i>UPPER(str)</i>	Переводит все символы строки <i>str</i> в верхний регистр

•

• Приложение 3. Основные функции даты и времени MySQL

- **Обозначение** **Описание** *ADDDATE(date, INTERVAL expr type)* Возвращает дату *date*, к которой прибавлен временной интервал, определяемый вторым параметром. Например,
- *ADDDATE('2009-03-20', INTERVAL 10 DAY)* *ADDTIME(expr1, expr2)* Возвращает результат сложения двух временных значений *CURDATE()* Возвращает текущую дату в формате 'YYYY-MM-DD' *CURTIME()* Возвращает текущее время суток в формате 'hh:mm:ss' *DATE(datetime)* Извлекает из значения *datetime* дату, отсекая часы, минуты и секунды *DATEDIFF(begin, end)* Вычисляет разницу в днях между датами *begin* и *end* *DATE\_FORMAT(date, format)* Форматирует время *date* в соответствии со строкой *format* *DAY(date)* Возвращает порядковый номер дня в месяце (от 1 до 31) *DAYNAME(date)* Возвращает день недели в виде полного английского названия *DAYOFWEEK(date)* Возвращает порядковый номер дня недели. В западных странах неделя начинается с воскресенья, номер которого 1. *DAYOFYEAR(date)* Возвращает порядковый номер дня в году (от 1 до 366) *EXTRACT(type FROM datetime)* Принимает дату и время суток и возвращает часть, определяемую параметром *type*. Например,
  - *EXTRACT(YEAR FROM '2009-12-31 14:30:15')* *FROM\_DAYS(N)* Принимает число дней *N*, прошедших с нулевого года, и возвращает дату в формате 'YYYY-MM-DD'. Обычно используется совместно с функцией *TO\_DAYS(date)*  *HOUR(datetime)* Извлекает из значения *datetime* часы (от 0 до 23) *LAST\_DAY(datetime)* Принимает дату и время суток и возвращает дату – последний день текущего месяца *MAKEDATE(year, dayofyear)* Принимает год *year* и номер дня в году *dayofyear* и возвращает дату в формате 'YYYY-MM-DD' *MAKETIME(hour, minute, second)* Принимает час *hour*, минуты *minute* и секунды *second* и возвращает время суток в формате 'hh:mm:ss' *MINUTE(datetime)* Извлекает из значения *datetime* минуты (от 0 до 59) *MONTH(datetime)* Возвращает числовое значение месяца года (от 1 до 12) *MONTHNAME(datetime)* Возвращает название месяца в виде полного английского названия *NOW()* Возвращает текущую дату и время в формате 'YYYY-MM-DD hh:mm:ss'
- *PERIOD\_ADD(period, N)*
- Добавляет *N* месяцев к значению даты *period*. Аргумент *period* должен быть представлен в числовом формате YYYYMMDD или YYMM
- 
- 
- Основные функции даты и времени MySQL (продолжение)
- Обозначение
- Описание
- 
- *PERIOD\_DIFF(period1, period2)*
- Вычисляет разницу в месяцах между двумя датами, представленными в числовом формате YYYYMMDD или YYMM
- 
- *QUARTER(datetime)*
- Возвращает значение квартала года (от 1 до 4)
- 
- *SECOND(datetime)*
- Извлекает из значения *datetime* секунды (от 0 до 59)
- 
- *SUBDATE(date, INTERVAL expr type)*
- Возвращает дату *date*, из которой вычитается временной интервал, определяемый вторым параметром. Например,

