ЛАБОРАТОРНАЯ РАБОТА №5
СОЗДАНИЕ БАЗЫ ДАННЫХ

Цель работы: изучить основы баз данных и научиться создавать базы данных в СУБД MySQL.

Ход выполнения лабораторной работы должен быть отражен в отчете. Отчет должен содержать титульный лист, номера заданий, коды программ, скриншот с результатом выполнения программы.

5.1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

5.1.1 Варианты хранения информации

Задача длительного хранения информации очень часто встречается в программировании web-приложений: подсчет посетителей в счетчике, хранение сообщений в форуме, удаленное управление содержанием информации на сайте и т. д. Существует ***два*** способа хранения данных: ***в текстовых файлах*** *и в* ***базах данных.***

При написании web-приложений на PHP первый способ настолько же прост, насколько ограничен.

Базы данных:

– сами заботятся о безопасности информации и ее сортировке;

– позволяют извлекать и размещать информацию при помощи одной строки;

– выборка информации из базы данных происходит значительно быстрее, чем из файла;

– код с использованием базы данных получается более компактным, и отлаживать его гораздо легче.

5.1.2 Классификация баз данных

База данных – это файл или набор файлов, хранящий структурированную определенным образом информацию. Для обработки этой информации используются особые программы, называемые ***системами управления базами данных.***

Задача длительного хранения и обработки информации появилась практически сразу с появлением первых компьютеров. Для решения этой задачи в конце 1960-х гг. были разработаны специализированные программы, получившие название *систем управления базами данных* (СУБД). СУБД проделали длительный путь эволюции. В конце 1980-х гг. доминирующей стала *система управления реляционными базами данных* (СУРБД). Для того чтобы унифицировать работу с ними, был разработан *структурированный язык запросов* (SQL), который представляет собой язык управления именно реляционными базами данных.

Взаимодействие с базой данных происходит при помощи СУБД, которая расшифровывает запросы и производит операции с информацией в базе данных. Поэтому более правильно было бы говорить о запросе к СУБД и о взаимодействии с СУБД из web-приложения. Но так как это несколько усложняет восприятие, далее мы будем использовать термин «базы данных» (БД), подразумевая при этом СУБД.

Существуют следующие ***разновидности*** баз данных:

– иерархические;
– реляционные;
– объектно-ориентированные;
– гибридные.

Иерархическая БД основана на древовидной структуре хранения информации. В этом смысле иерархические базы данных очень напоминают файловую систему компьютера.

В ***реляционных*** БД информация собрана в таблицы, состоящие из столбцов и строк, на пересечении которых расположены ячейки. Запросы к таким базам данных возвращают таблицу, которая повторно может участвовать в следующем запросе. Данные в одних таблицах, как правило, связаны с данными других таблиц, откуда и произошло название «реляционные» (от англ. *relational* – родственный).

В ***объектно-ориентированных*** БД данные хранятся в виде объектов. С объектно-ориентированными базами данных удобно работать, применяя объектно-ориентированное программирование. Однако в настоящее время такие базы данных еще не достигли популярности реляционных, поскольку пока значительно уступают им в производительности.

Гибридные СУБД совмещают в себе возможности реляционных и объектно-ориентированных баз данных.

В web-приложениях, как правило, используются ***реляционные базы данных.***

5.1.3 Терминология реляционных баз данных

Кратко особенности реляционной базы данных можно описать следующим образом:

– данные хранятся в таблицах, состоящих из столбцов и строк;
– на пересечении каждого столбца и строки стоит в точности одно значение;
– у каждого столбца есть свое имя, которое служит его названием, и все значения в одном столбце имеют один тип;
– столбцы располагаются в определенном порядке, который определяется при создании таблицы, в отличие от строк, которые располагаются в произвольном порядке. В таблице может не быть ни одной строчки, но обязательно должен быть хотя бы один столбец.

Запросы к базе данных возвращают результат в виде таблиц, которые тоже могут выступать как объект запросов.

Таблицы в реляционных базах данных состоят из записей, а записи, в свою очередь, состоят из полей.

Поле (field) – это базовый элемент данных в БД. Поле имеет имя и тип.

Запись (record) – это набор полей, содержащих связанную информацию.

Таблица (table) – это набор записей одной структуры полей.

База данных (database) – это совокупность связанных таблиц.

Индекс (index) – это отсортированный список значений полей, предназначенный для ускорения поиска в базе данных. Обычно поиск производится по значению одного поля или по значению нескольких полей.

Уникальный индекс (index) – список значений поля, в котором каждое значение уникально, т. е. в одной таблице не должно быть два поля (по которому строился уникальный индекс) с одним и тем же значением.

Первичный ключ (primary key) представляет собой один из примеров уникальных индексов и применяется для уникальной идентификации записей таблицы. Никакие из двух записей таблицы не могут иметь одинаковые значения первичного ключа.

По способу задания первичных ключей различают ***логические*** (естественные) и ***суррогатные*** (искусственные) ключи. Для логического задания первичного ключа нужно выбрать в базе данных то, что естественным образом определяет запись (например, номер паспорта). Если подходящих примеров для естественного задания первичного ключа не находится, пользуются суррогатным ключом. Суррогатный ключ представляет собой дополнительное поле в базе данных, предназначенное для обеспечения записей первичным ключом.

Даже если в базе данных содержится естественный первичный ключ, лучше использовать суррогатные ключи, поскольку их применение позволяет абстрагировать первичный ключ от реальных данных. В этом случае облегчается работа с таблицами, поскольку суррогатные ключи не связаны ни с какими фактическими данными этой таблицы.

Запрос (query) – оператор, выбирающий записи и поля из одной или нескольких таблиц. При этом выбираемые записи и поля должны соответствовать условию, заданному оператором.

Схемой БД называется структура связей между полями и таблицами.

Нормализация схемы БД – это процедура, производимая над базой данных с целью удаления в ней избыточности. В нормализованной БД уменьшается вероятность возникновения ошибок, она занимает меньше места на жестком диске и т. д.

5.1.4 Настольные и сетевые реляционные СУБД

Как правило, любая СУБД состоит из двух частей. Первая часть – это та программа, с которой работает пользователь, – *клиент данных*. Вторая же часть непосредственно занимается базой данных: принимает от клиента запросы на выборку и изменение данных, выполняет их и возвращает клиенту. Это так называемый *процессор данных*. Можно сказать, что клиент данных занимается приемом запросов от пользователя и выводом результатов, а процессор – собственно обработкой данных.

И в зависимости от того, как реализованы клиент и процессор данных, СУБД делятся на две большие группы: ***настольные*** и ***клиент-серверные***.

Настольная СУБД реализована в виде одной-единственной программы; и клиент, и процессор данных слиты воедино в одном исполняемом файле. Это первое отличие. Второе отличие – настольная СУБД работает непосредственно с файлами баз данных точно так же, как Microsoft Word работает с файлами документов.

Серверная СУБД – это процессор данных, оформленный в виде отдельной программы и работающий на специально выделенном для этого серверном компьютере. Как и любой другой сервер, он принимает от клиентов запросы, считывает из файла базы данных, обрабатывает их и пересылает результаты обработки клиентам.

Преимущества настольных СУБД:

– исключительная легкость установки и использования;

– нетребовательность к дополнительному программному обеспечению.

Недостатки настольных СУБД:

– невысокое быстродействие при многопользовательском доступе к базе данных по сети;

– недостаточная надежность и защищенность.

Поэтому настольные СУБД используются для ведения персональных баз данных и для создания совсем небольших, как правило, несетевых систем обработки данных.

Преимущества серверных СУБД:

– большая производительность (поскольку по сети пересылаются только запросы и ответы, которые меньше по размерам, чем фрагменты файлов);

– большая надежность и защищенность.

Недостатки серверных СУБД: сложность установки, настройки и сопровождения.

Настольные СУБД – это Microsoft Access, Corel Paradox, Borland dBase, Microsoft FoxPro и др.

К серверным СУБД относятся Borland InterBase, MySQL, Microsoft SQL Server, PostgreSQL, Informix, Oracle, Sybase, IBM DB2 и многое другое.

5.1.5 Архитектура web-баз данных

Базовая архитектура web-баз данных включает в себя web-браузер, web-сервер, сценарный механизм и сервер баз данных (рисунок 5.1).

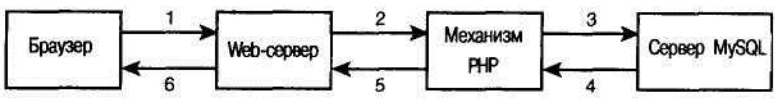

Рисунок 5.1 – Архитектура web-баз данных

Типичная транзакция web-базы данных состоит из этапов, обозначенных цифрами на рисунке 5.1.

1 Web-браузер пользователя отправляет HTTP-запрос определенной web-страницы. Например, поиск в магазине всех книг, написанных определенным автором, используя HTML-форму. Страница с результатами поиска называется results.php.

2 Web-сервер принимает запрос на results.php, получает файл и передает его механизму PHP на обработку.

3 Механизм PHP начинает синтаксический анализ сценария. В сценарии присутствует команда подключения к базе данных и выполнения запроса в ней (поиск книг). PHP открывает соединение с сервером MySQL и отправляет необходимый запрос.

4 Сервер MySQL принимает запрос в базу данных, обрабатывает его, а затем отправляет результаты (в данном случае – список книг) обратно в механизм PHP.

5 Механизм PHP завершает выполнение сценария, форматируя результаты запроса в виде HTML, после чего отправляет результаты в HTML-формате web-серверу.

6 Web-сервер пересылает HTML в браузер, с помощью которого пользователь просматривает список необходимых книг.

Процесс этот, как правило, протекает вне зависимости от того, какой сценарный механизм и какой сервер баз данных используется. Зачастую программное обеспечение web-сервера, механизм PHP и сервер баз данных находятся в одной машине. Правда, не менее часто сервер базы данных работает на другой машине. Это делается из соображений безопасности, увеличения объема или разделения потока. С точки зрения перспектив развития, в работе оба варианта одинаковы, однако в плане производительности второй вариант может оказаться более предпочтительным.

5.1.6 СУБД MySQL

Одна из самых популярных СУБД, которые используются в web-программировании, – MySQL. Она предназначена для создания сравнительно небольших (в районе 100 Мбайт) баз данных и поддерживает некоторое подмножество языка запросов SQL.

Достоинства MySQL:

– MySQL – весьма быстрый и нетребовательный к ресурсам компьютера сервер данных;

– возможностей MySQL более чем хватает для создания сравнительно небольших web-сайтов;

– MySQL распространяется бесплатно, более того – его исходные тексты открыты для изучения и доработки;

– MySQL прекрасно работает в связке с PHP, технологией создания активных серверных web-страниц;

MySQL – это программа-сервер, постоянно работающая на компьютере. Клиентские программы (например, сценарии) посылают ей специальные запросы через механизм сокетов (т. е. при помощи сетевых средств), она их обрабатывает и запоминает результат. Затем также по специальному запросу клиента весь этот результат или его часть передается обратно.

Один сервер MySQL может поддерживать сразу несколько баз данных, доступ к которым может разграничиваться именем пользователя и паролем.

5.1.7 Типы данных, используемые в базе данных MySQL

В MySQL определены три базовых типа столбцов: числовой, дата и время и строковый. Каждая из этих категорий подразделяется на множество типов. Каждый из типов обладает различной емкостью. Выбирая тип столбца, главное – выбрать наименьший, в котором помещаются данные.

5.1.7.1 Целые числа

Общий вид указания типа данных:

***префикс*INT [(M)] [UNSIGNED]**

Необязательный флаг UNSIGNED указывает, что будет создано поле для хранения беззнаковых чисел (больших или равных 0). При объявлении целого типа можно указать максимальную ширину отображения. В приведенных ниже таблицах для типов данных этот параметр обозначается как ***M***. Если для данного типа он не обязателен, то он приводится в квадратных скобках. Максимальным значением ***M*** является 255, минимальным – 1. Параметр ***M*** определяет только количество цифр, которое будет выводиться при запросе в клиентах типа MySQL, и не влияет на размер памяти, отводимой для хранения значения (таблица 5.1).

Таблица 5.1 – Целые типы данных

Название типа	Диапазон значений	Размер
TINYINT	от -128 до 127 (от -2^7 до 2^7-1), от 0 до 255 (от 0 до 2^8-1)	1 байт
SMALLINT	от -2^{15} до $2^{15}-1$, от 0 до $2^{16}-1$	2 байта
MEDIUMINT	от -2^{23} до $2^{23}-1$, от 0 до $2^{24}-1$	3 байта
INT	от -2^{23} до $2^{23}-1$, от 0 до $2^{32}-1$	4 байта
BIGINT	от -2^{63} до $2^{63}-1$, от 0 до $2^{64}-1$	8 байт

5.1.7.2 Дробные числа

Точно так же, как целые числа подразделяются в MySQL на несколько разновидностей, MySQL поддерживает и несколько типов дробных чисел (таблица 5.2). В общем виде они записываются так:

***ИмяТипа*[(length, decimals)] [UNSIGNED]**

где *length* – количество знакомест (ширина поля), в которых будет размещено дробное число при его передаче;

decimals – количество знаков после десятичной точки, которые будут учитываться;

UNSIGNED – задает беззнаковые числа.

Таблица 5.2 – Дробные типы данных

Название типа	Описание	Размер
FLOAT	Число с плавающей точкой небольшой точности	4 байта
DOUBLE	Число с плавающей точкой двойной точности	8 байт
REAL	Синоним для DOUBLE	8 байт
DECIMAL	Дробное число, хранящееся в виде строки	length + 2 байта
NUMERIC	Синоним для DECIMAL	length + 2 байта

5.1.7.3 Строки

Строки представляют собой массивы символов. Строковые типы разделяются на три группы.

Первая группа простые старые строки, которые представляют собой короткие фрагменты текста (таблица 5.3).

Таблица 5.3 – Строковые типы данных (первая группа)

Название типа	Описание	Размер
CHAR (M) [BINARY]	Символы с фиксированной длиной. От 1 до 255 символов	Объем памяти M байт. Максимальный размер M байт
VARCHAR(M) [BINARY]	Символы с произвольной длиной. От 1 до 255 символов	Объем памяти L+1 байт. Максимальный размер M байт

Если указан флаг BINARY, то при запросе SELECT строка будет сравниваться с учетом регистра.

Ширину каждого из них можно регулировать. Столбцы с типом **CHAR** будут дополняться пробелами до максимальной ширины, независимо от размеров данных, в то время как в столбцах с типом **VARCHAR** ширина зависит от размеров данных. MySQL усекает пробелы в конце текстовых строк у **CHAR** во время извлечения и у **VARCHAR** во время сохранения.

Вторая группа – это типы **TEXT** и **BLOB** (таблица 5.4). Их размеры могут быть разными. Первый тип данных предназначен для более длинных текстовых фрагментов, второй – для двоичных. Поля типа BLOB (Binary Large Object – большой двоичный объект) могут содержать любые данные, в том числе звуки и изображения. На практике столбцы **TEXT** и **BLOB** одинаковы, за исключением того, что **TEXT** чувствителен к регистру, а **BLOB** – нет.

Таблица 5.4 – Строковые типы данных (вторая группа)

Название типа	Описание
TINYTEXT	Может хранить не более 255 символов
TINYBLOB	Может хранить не более 255 символов
TEXT	Может хранить не более 65 535 символов
BLOB	Может хранить не более 65 535 символов
MEDIUMTEXT	Может хранить не более 16 777 215 символов
MEDIUMBLOB	Может хранить не более 16 777 215 символов
LONGTEXT	Может хранить не более 4 294 967 295 символов
LONGBLOB	Может хранить не более 4 294 967 295 символов

К *третьей группе* принадлежат два специальных типа **SET** и **ENUM**. Тип **SET** предназначен для того, чтобы определять, что значения в данном столбце принадлежат конкретному набору фиксированных значений. Значения столбца могут содержать несколько значений из набора. В определении набора можно задать вплоть до 64 элементов.

ENUM представляет собой перечисление. Этот тип очень похож на SET, но столбцы этого типа могут иметь всего лишь одно из фиксированных значений или NULL, а максимальное количество элементов в перечислении составляет 65 535.

ENUM(‘value1’, ‘value2’, …)
SET(‘value1’, ‘value2’, …)

5.1.7.4 Дата и время

MySQL поддерживает несколько типов полей, специально приспособленных для хранения дат и времени в различных форматах (таблица 5.5).

Таблица 5.5 – Типы данных даты и времени

Название типа	Описание	Размер
DATE	Дата в формате ГГГГ-ММ-ДД	3 байта
TIME	Время в формате ЧЧ:ММ:СС	3 байта
DATETIME	Дата и время в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС	8 байт
TIMESTAMP [(M)]	Дата и время в формате timestamp. Метка времени, полезная для отчетов по транзакциям. Формат отображения зависит от значения M	4 байта
YEAR[(2|4)]	Год. Можно определить двух- или четырехциферный формат	–

5.1.8 Команды MySQL

Сервер MySQL поддерживает как инструкции SQL, так и служебные команды MySQL, предназначенные для администрирования и использования таблиц в базах данных MySQL.

Язык структурированных запросов SQL (Structure Query Language) позволяет выполнять различные операции с базами данных:

- создавать базы данных и таблицы;
- добавлять информацию в таблицы;
- удалять информацию;
- модифицировать информацию;
- получать нужные данные.

Команда **CREATE DATABASE** создает новую базу данных:

CREATE DATABASE [IF NOT EXISTS] *db_name*;

Здесь *db_name* является именем создаваемой базы данных. Не обязательная ключевая фраза IF NOT EXISTS сообщает, что базу данных следует создавать, только если база данных с таким именем отсутствует, что позволяет предотвратить завершение запроса ошибкой. Особенно это актуально при использовании SQL в PHP-скриптах.

Пример 5.1

Создание базы данных forum:

```
CREATE DATABASE db_test;
```

Для **удаления** базы данных используется команда **DROP DATABASE**. Синтаксис:

```
DROP DATABASE database_name;
```

где *database_name* – задает имя базы данных, которую необходимо удалить.

Пример 5.2

Удаление базы данных db_test:

```
DROP DATABASE db_test;
```

Задание 5.1:

5.1.1 Запустить web-сервер Apache.

5.1.2 Запустить браузер и в адресной строке ввести следующий адрес: http://localhost/Tools/phpMyAdmin/.

phpMyAdmin *– web-приложение с открытым кодом, написанное на языке PHP и представляющее собой web-интерфейс для администрирования СУБД MySQL. phpMyAdmin позволяет через браузер осуществлять администрирование сервера MySQL, запускать команды SQL и просматривать содержимое таблиц и баз данных.*

5.1.3 Создать базу данных, которая хранит сообщения форума. Структура форума может быть следующей. Имеется список разделов, переход по которым приводит посетителя к списку тем раздела. При переходе по теме посетитель приходит к обсуждению этой темы, состоящему из сообщений других посетителей.

Для создания базы данных в phpMyAdmin необходимо ввести имя базы данных, выбрать кодировку и нажать кнопку создать (рисунок 5.2).

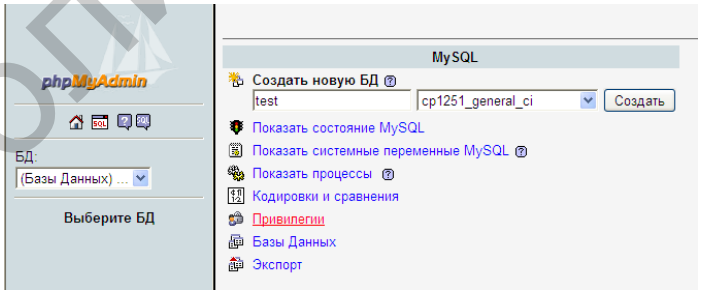

Рисунок 5.2 – Создание базы данных в phpMyAdmin

Созданная база данных отобразится в списке (рисунок 5.3).

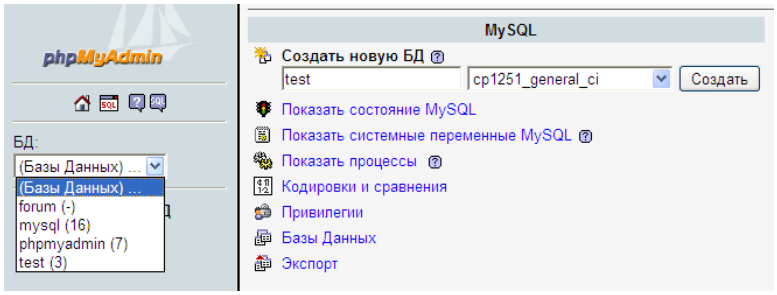

Рисунок 5.3 – Просмотр списка созданных баз данных в phpMyAdmin

5.1.4 Сделать созданную базу данных активной, выбрав ее имя в списке баз данных. Перейти на закладку SQL. Все дальнейшие задания будут выполняться путем ввода SQL команд (рисунок 5.4).

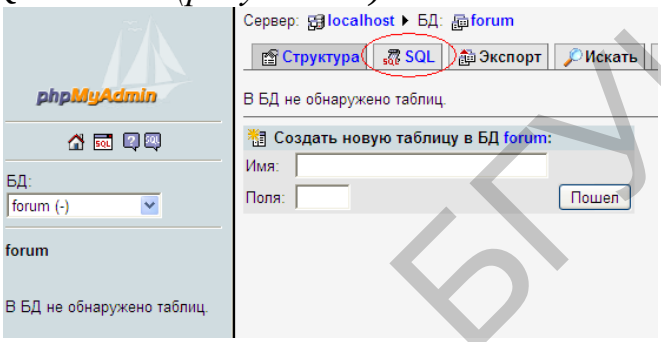

Рисунок 5.4 – Выбор меню для ввода SQL-запросов в phpMyAdmin

Команда **CREATE TABLE** создает новую таблицу в выбранной базе данных. В простейшем случае команда имеет следующий синтаксис:

CREATE TABLE *table_name*(*column_name1 type, column_name2 type*,...)
[table_options];

Здесь *table_name* – имя новой таблицы; *column_name* – имена колонок (полей), которые будут присутствовать в создаваемой таблице; *type* – определяет тип создаваемой колонки; *table_options* – необязательное указание типа таблицы, например TYPE = MyISAM.

Пример 5.3

Создание таблицы телефонных номеров друзей, которая будет состоять из трех столбцов: ФИО друга, адрес и телефон:

CREATE TABLE tel_numb(fio text, address text, tel text);

К типу данных можно присоединить модификаторы, которые задают его свойства и те операции, которые можно (или, наоборот, запрещено) выполнять с соответствующими столбцами.

Not null – означает, что поле не может содержать неопределенное значение, т. е. поле обязательно должно быть инициализировано при вставке новой записи в таблицу (если не задано значение по умолчанию).

Пример 5.4

Для таблицы с телефонами нужно указать, что поле с ФИО друга (поле fio) и его телефоном (поле tel) не может иметь неопределенное значение:

```
CREATE TABLE tel_numb(fio text NOT NULL, address text, tel text NOT NULL);
```

Primary key – отражает, что поле является первичным ключом, т. е. идентификатором записи, на который можно ссылаться.

Пример 5.5

```
CREATE TABLE tel_numb(fio text, address text, tel text, PRIMARY KEY (fio));
```

Auto_increment – при вставке новой записи поле получит уникальное значение, так что в таблице никогда не будут существовать два поля с одинаковыми номерами.

Пример 5.6

```
CREATE TABLE tel_numb(id int AUTO_INCREMENT, fio text, tel text);
```

Default – задает значение по умолчанию для поля, которое будет использовано, если при вставке записи для этого поля не было явно указано значение.

Пример 5.7

```
CREATE TABLE tel_numb(fio text, address text DEFAULT 'Не указан', tel text)
```

Задание 5.2:

5.2.1 Создать первую таблицу базы данных forum, которая называется authors и содержит различные данные о зарегистрированных посетителях форума: имя (name), пароль (passw), e-mail (email), web-адрес сайта посетителя (url), номер ICQ (icq), сведения о посетителе (about), строку, содержащую путь к файлу фотографии посетителя (photo), время добавления запроса (time), последнее время посещения форума (lasttime), статус посетителя – является ли он модератором, администратором или обычным посетителем (statususer). Кроме перечисленных полей в таблице имеется поле id_author, являющееся первичным ключом таблицы.

```
CREATE TABLE authors (
    id_author INT(6) NOT NULL AUTO_INCREMENT,
    name TINYTEXT,
    passw TINYTEXT,
    email TINYTEXT,
    url TINYTEXT,
    icq TINYTEXT,
    about TINYTEXT,
    photo TINYTEXT,
    time DATETIME DEFAULT NULL,
```

```
    last_time DATETIME DEFAULT NULL,
    themes INT(10) DEFAULT NULL,
    statususer INT(2) DEFAULT NULL,
    PRIMARY KEY (id_author)
) TYPE=MyISAM;
```

5.2.2 Создать вторую таблицу базы данных forum, которая содержит данные о разделе форума и называется forums. В таблице forums присутствуют следующие поля: первичный ключ (idforum), название раздела (name), правила форума (rule), краткое описание форума (logo), порядковый номер (pos), флаг, принимающий значение 1, если форум скрытый, и 0, если общедоступный (hide).

```
CREATE TABLE forums (
    id_forum INT(6) NOT NULL AUTO_INCREMENT,
    name TINYTEXT,
    rule TEXT,
    logo TEXT,
    pos INT(6) DEFAULT NULL,
    hide TINYINT(1) DEFAULT NULL,
    PRIMARY KEY (id_forum)
) TYPE=MyISAM;
```

5.2.3 Создать таблицу themes, содержащую список тем. В таблице присутствуют следующие поля: первичный ключ (id_theme); название темы (name); автор темы (author); внешний ключ к таблице authors (id_author); флаг, принимающий значение 1, если тема скрытая, и 0, если тема отображается (hide), – это поле необходимо для моделирования; время добавления темы (time); внешний ключ к таблице форумов (id_forum), чтобы определить, к какому разделу форума относится данная тема.

В таблице themes нормализация проведена частично, она содержит два внешних ключа: id_author и id_forum – для таблиц посетителей и списка форумов, в то же время в ней дублируется имя автора author, присутствующее в таблице посетителей authors под именем name. Этот случай служит примером денормализации, предназначенной для того, чтобы не запрашивать каждый раз имена из таблицы authors при выводе списка тем и их авторов и обеспечить приемлемую скорость работы форума.

```
CREATE TABLE themes (
    id_theme INT(11) NOT NULL AUTO_INCREMENT,
    name TEXT,
    author TEXT,
    id_author INT(6) DEFAULT NULL,
    hide TINYINT(1) DEFAULT NULL,
    time DATETIME DEFAULT NULL,
    id_forum TINYINT(2) DEFAULT NULL,
    PRIMARY KEY (id_theme)
) TYPE=MyISAM;
```

5.2.4 Создать таблицу posts, в которой хранятся сообщения посетителей. В таблице присутствуют следующие поля: первичный ключ (id_post); тело сообщения (name); необязательная ссылка на ресурс, которую автор сообщения может ввести при добавлении сообщения (url); путь к файлу, прикрепляемому к сообщению (file); имя автора (author); внешний ключ к таблице authors (id_author); флаг (hide), принимающий значение 1, если сообщение отмечено как скрытое, и 0, если оно отображается, – необходим для моделирования; время добавления сообщения (time); сообщение, ответом на которое является данное сообщение (parent_post), это поле равно 0, если данное сообщение – первое по этой теме; внешний ключ к таблице тем (id_theme), для того чтобы определить, к какой теме относится сообщение.

```
CREATE TABLE posts (
    id_post INT(11) NOT NULL AUTO_INCREMENT,
    name TEXT,
    url TINYTEXT,
    file TINYTEXT,
    author TINYTEXT,
    id_author INT(6) DEFAULT NULL,
    hide TINYINT(1) DEFAULT NULL,
    time DATETIME DEFAULT NULL,
    parent_post INT(11) DEFAULT NULL,
    id_theme int(11) DEFAULT NULL,
    PRIMARY KEY (id_post)
) TYPE=MyISAM;
```

Команда **SHOW TABLES** предназначена для просмотра списка таблиц в текущей базе данных.

Задание 5.3. *Убедитесь, что все таблицы успешно созданы, выполнив команду SHOW TABLES.*

Команда **DESCRIBE** показывает структуру созданных таблиц и имеет следующий синтаксис:

```
DESCRIBE table_name;
```

где *table_name* – имя таблицы, структура которой запрашивается.

Команда DESCRIBE не входит в стандарт SQL и является внутренней командой СУБД MySQL.

Пример 5.8

Просмотр структуры таблицы tel_numb с выполнением команды:

```
DESCRIBE tel_numb;
```

Задание 5.4. *Проверьте правильность создания таблиц, посмотрев структуру каждой таблицы (выполнять каждую команду необходимо по отдельности):*

DESCRIBE authors;
DESCRIBE forums;
DESCRIBE posts;
DESCRIBE themes;

Переименование таблицы (ALTER TABLE RENAME). Переименование таблицы можно сделать при помощи следующей конструкции:

ALTER TABLE *table_name_old* RENAME *table_name_new;*

где *table_name_old* – старое имя таблицы, которое нам нужно переименовать; *table_name_new* – новое имя таблицы.

Пример 5.9

Переименование таблицы *search* в *search_en*:

ALTER TABLE search RENAME search_en;

Вставка столбцов (ALTER TABLE ADD). Вставку нового столбца можно осуществить при помощи следующей конструкции:

ALTER TABLE *table_name* ADD *field_name parameters;*

где *table_name* – имя таблицы, в которой будет вставлен новый столбец; *field_name* – имя вставляемого столбца; *parametrs* – параметры, описывающие вставляемый столбец. Обязательным параметром является указание типа данных.

Пример 5.10

Вставка в таблицу *my_friends* столбца под названием *adress_2*, который будет содержать текстовые значения:

ALTER TABLE my_friends ADD adress_2 TEXT;

По умолчанию новый столбец вставляется в конец таблицы. Если необходимо, чтобы столбец встал в начало таблицы, нужно после параметров вставляемого столбца написать ключевое слово **FIRST**:

ALTER TABLE my_friends ADD adress_2 TEXT FIRST;

Если необходимо, чтобы столбец был вставлен не в начале таблицы и не в конце, а после определенного столбца, то нужно применить ключевое слово **AFTER** *имя столбца*, после которого будет установлен новый столбец:

ALTER TABLE my_friends ADD adress_2 TEXT AFTER adress_1;

В этом примере новый столбец *adress_2* будет установлен после столбца *adress_1*.

Если нужно дописать к таблице не один, а несколько столбцов, то для каждого столбца нужно **ADD** *field_name parametrs* записать через запятую:

ALTER TABLE my_friends ADD adress_2 TEXT,
ADD adress_3 TEXT, ADD adress_4 TEXT;

В случае если надо дописать два столбца внутри таблицы, можно поступить следующим образом:

ALTER TABLE my_friends ADD adress_2 TEXT AFTER adress_1,
ADD adress_3 TEXT AFTER adress_2;

т. е. первый вставляемый столбец записываем после *adress_1*, а второй – после первого.

Изменение свойств столбца (ALTER TABLE CHANGE). Изменить свойства одного или нескольких столбцов можно при помощи следующей конструкции:

ALTER TABLE *table_name* CHANGE *field_name_old field_name_new parametrs;*

где *table_name* – имя таблицы, в которой находится изменяемый столбец; *field_name_old* – имя столбца изменяемого столбца; *field_name_new* – новое имя изменяемого столбца (должно равняться *field_name_old*, если мы не хотим поменять имя столбца); *parametrs* – новые параметры столбца.

Пример 5.11

Установление типа строки field_1 как текст:

ALTER TABLE my_table CHANGE field_1 field_1 TEXT;

А если необходимо при этом еще и переименовать столбец в field_2, то получится так:

ALTER TABLE my_table CHANGE field_1 field_2 TEXT;

В случае, если надо изменить свойства сразу нескольких столбцов, то конструкцию **CHANGE** *field_name_old field_name_new parametrs* повторяем через запятую для каждого столбца:

ALTER TABLE my_table CHANGE field_1 field_2 TEXT,
CHANGE field_3 field_3 TEXT";

Удаление столбцов (ALTER TABLE DROP). Удаление столбца можно сделать при помощи следующей конструкции:

ALTER TABLE table_name DROP field_name

где *table_name* – имя таблицы, в которой будет удален столбец; *field_name* – имя удаляемого столбца.

Пример 5.12

ALTER TABLE search DROP id_num;

Если мы хотим удалить сразу несколько полей, то надо через запятую повторить **DROP** *field_name* для каждого столбца:

ALTER TABLE search DROP id_1, DROP id_2, DROP id_3;

Задание 5.5:

5.5.1 Добавить в таблицу forums новый столбец test, разместив его после столбца name.

ALTER TABLE forums ADD test INT(10) AFTER name;

5.5.2 Посмотреть структуру измененной таблицы forums.

DESCRIBE forums;

5.5.3 Переименовать созданный столбец test в текстовый столбец new_test.

ALTER TABLE forums CHANGE test new_test TEXT;

5.5.4 Посмотреть структуру измененной таблицы forums.

DESCRIBE forums;

5.5.5 Изменить тип столбца new_test на целочисленный. Установить, что значение этого столбца не может быть пустым. При изменении только типа столбца, а не его имени, указание имени необходимо, хотя в этом случае оно будет фактически повторяться.

ALTER TABLE forums CHANGE new_test new_test INT(5) NOT NULL;

5.5.6 Посмотреть структуру измененной таблицы forums.

DESCRIBE forums;

5.5.7 Удалить созданный столбец new_test. Посмотреть структуру измененной таблицы.

ALTER TABLE forums DROP new_test;

5.5.8 Посмотреть структуру измененной таблицы forums.

DESCRIBE forums;

Команда **INSERT INTO .. VALUES** вставляет новые записи в существующую таблицу. Синтаксис команды:

INSERT INTO table_name(field_name1, field_name2,...) values('content1', 'content2',...)

Данная команда добавляет в таблицу *table_name* запись, у которой поля, обозначенные как *field_nameN*, установлены в значение *contentN*.

Пример 5.13

Добавляем записи в таблицу адресов и телефонов (ФИО, адрес, телефон):

INSERT INTO tel_numb(fio, address, tel)
values('Вася Пупкин', 'ул.Горького, д.18', '23-23-23')

Те поля, которые не были перечислены в команде вставки, получают «неопределенные» значения (неопределенное значение – это не пустая строка, а просто признак, который «говорит» MySQL, что у данного поля нет никакого значения).

Если при создании таблицы поле было отмечено флажком NOT NULL и оно при вставке записи получило неопределенное значение, то MySQL возвратит ошибку.

Команда **DELETE** предназначена для удаления записей из таблицы:

DELETE FROM *table_name* WHERE (*выражение*);

Данная команда удаляет из таблицы *table_name* все записи, для которых выполнено *выражение*. *Выражение* – это просто логическое выражение.

Пример 5.14

Удаление записи из таблицы, содержащей ФИО, адрес и телефон:

DELETE FROM tel_numb WHERE (fio='Вася Пупкин');

или удаление по нескольким параметрам

DELETE FROM tel_numb WHERE (fio='Вася Пупкин' AND tel='23-45-45');

В выражении помимо имен полей, констант и операторов могут также встречаться простейшие вычисляемые части, например (id<10+4*5).

Обновление записи осуществляется командой **UPDATE**:

UPDATE *table_name* SET *field_name1='var1', field_name2='var2',...* [WHERE (*выражение*)] [LIMIT *rows*)];

Данная команда для всех записей в таблице *table_name*, удовлетворяющих выражению *выражение*, устанавливает указанные поля *field_nameN* в значении *varN*. В выражении WHERE, если оно присутствует, задается, какие строки подлежат обновлению. В остальных случаях обновляются все строки. Ключевое слово LIMIT позволяет ограничить число обновляемых строк.

Эту команду удобно применять, если не требуется обновлять не все поля какой-то записи, а нужно обновить только некоторые.

Пример 5.15

Если необходимо, для записей таблицы, у которых поле C_NO = 1 – это код клиента Иванова, установить значение CITY равным "Псков"

UPDATE CLIENTS SET CITY = 'Псков' WHERE C_NO = 1;

Выборка данных. Оператор **SELECT** является краеугольным камнем всего языка SQL. Он используется, чтобы выполнить запросы к базе данных. Это действительно основа языка SQL. Синтаксис оператора:

```
SELECT [STRAIGHT_JOIN] [DISTINCT | ALL] select_expression,.
  [FROM tables... [WHERE where_definition] [GROUP BY column,...]
  [ORDER BY column [ASC | DESC], ...]
```

Как видно из вышеприведенного, вместе с командой SELECT используются ключевые слова, использование которых очень влияет на ответ сервера. Рассмотрим каждое из них.

DISTINCT. Пропускает строки, в которых все выбранные поля идентичны, т. е. устраняет дублирование данных.

WHERE. Предложение команды SELECT, которое позволяет устанавливать предикаты, условие которых может быть верным или неверным для любой строки таблицы. Извлекаются только те строки, для которых такое утверждение верно.

Пример 5.16

Запрос выводит колонки u_id и lname из таблицы publishers, для которых значение в столбце city – New York. Это дает возможность сделать запрос более конкретным.

SELECT u_id, lname FROM publishers WHERE city ='New York';

IN. Оператор IN определяет набор значений, в котором данное значение может или не может быть включено.

Пример 5.17

Например, запрос

```
SELECT * FROM Salespeople WHERE city = 'Barcelona' OR city = 'London';
```

может быть переписан более просто:

```
SELECT * FROM Salespeople WHERE city IN ( 'Barcelona', 'London' );
```

IN определяет набор значений с помощью имен членов набора, заключенных в круглые скобки и отделенных запятыми. Затем он проверяет различные значения указанного, пытаясь найти совпадение со значениями из набора. Если это случается, то предикат верен. Когда набор содержит значения номеров, а не символов, одиночные кавычки опускаются.

Символ * в операторе SELECT предписывает, что из отобранных записей следует извлечь *все* поля, когда будет выполнена команда получения выборки. С другой стороны, вместо звездочки можно через запятую непосредственно перечислить имена полей, которые требуют извлечения.

BETWEEN. Оператор BETWEEN похож на оператор IN. В отличие от определения по номерам из набора, как это делает IN, BETWEEN определяет диапазон, значения которого должны уменьшаться, что делает предикат верным. Вы должны ввести ключевое слово BETWEEN с начальным значением, ключевое AND и конечное значение. В отличие от IN, BETWEEN чувствителен к порядку, и первое значение в предложении должно быть первым по алфавитному или числовому порядку.

Пример 5.18

```
SELECT * FROM Salespeople WHERE comm BETWEEN .10 AND .12;
SELECT * FROM Salespeople WHERE city BETWEEN 'Berlin' AND 'London';
```

LIKE. LIKE применим только к полям типа CHAR или VARCHAR, с которыми он используется чтобы находить подстроки. То есть он ищет поле символа, чтобы видеть, совпадает ли с условием часть его строки. В качестве условия он использует групповые символы (wildkards) – специальные символы, которые могут соответствовать чему-нибудь. Имеются два типа групповых символов, используемых с LIKE:

- символ подчеркивания (_) замещает любой одиночный символ;
- знак '%', замещающий любое количество символов.

Пример 5.19

Если заданы следующие условия:

```
SELECT * FROM Customers WHERE fname LIKE 'J%';
```

то будут выбраны все заказчики, чьи имена начинаются на J: John, Jerry, James и т. д.

COUNT. Агрегатная функция, производит подсчет значений в столбце или числа строк в таблице. При работе со столбцом использует DISTINCT в качестве аргумента.

Пример 5.20

```
SELECT COUNT ( DISTINCT snum ) FROM Orders;
```

Пример 5.21

При подсчете строк имеет синтаксис:

```
SELECT COUNT (*) FROM Customers;
```

GROUP BY. Предложение GROUP BY позволяет определять подмножество значений в особом поле в терминах другого поля и применять функцию агрегата к подмножеству. Это дает возможность объединять поля и агрегатные функции в едином предложении SELECT.

Пример 5.22

Запрос позволяет найти наибольшую сумму приобретений, полученную каждым продавцом.

```
SELECT snum, MAX (amt) FROM Orders GROUP BY snum;
```

В предложении GROUP могут быть использованы следующие функции:

- AVG() – среднее для группы GROUP;
- SUM() – сумма элементов GROUP;
- COUNT() – число элементов в GROUP;
- MIN () – минимальный элемент в GROUP;
- MAX() – максимальный элемент в GROUP.

Здесь MIN() и MAX() могут принимать строку или число в качестве аргумента. Эти функции не могут использоваться в выражении, хотя их параметр может быть выражением.

ORDER BY. Эта команда упорядочивает вывод запроса согласно значениям в том или ином количестве выбранных столбцов. Многочисленные столбцы упорядочиваются один внутри другого так же, как с GROUP BY.

5.2 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Создать базу данных для книжного магазина BookBy. В базе данных необходимо хранить данные о клиентах, о продаваемых книгах и об особенностях заказов.

Исходя из этого, делаем вывод, что в базе данных необходимы, как минимум, три таблицы: Customers (Клиенты), Orders (Заказы) и Books (Книги). Схема приложения имеет следующий вид (*далее приведен пример записей, которые необходимо добавить в базу данных, что поможет определить тип каждого поля*):

Customers (CustomerlD, Name, Address, City)
Orders (OrderlD, CustomerlD, Amount, Date)
Books (ISBN, Author, Title, Price)
Order_ltems (OrderlD, ISBN, Quantity)
Book_Reviews (ISBN, Reviews)
Первичные ключи подчеркивают обычной линией.

Добавить в базу данных ***bookby*** по 2-3 записи в каждую таблицу, используя командную строку. Например:

```
INSERT INTO customers VALUES
    (NULL, "Иванов И.И.", "пр. Пушкина 32", "Минск"),
    (NULL, "Петров И.В.", "пр. Рокоссовского 10", "Минск"),
    (NULL, "Сидоров А.В.", "ул. Цветочная 18", "Пинск");
INSERT INTO orders VALUES
    (NULL, 3, 69.98, "02-Apr-2006"),
    (NULL, 1, 49.99, "15-Apr-2006"),
    (NULL, 2, 74.98, "19-Apr-2006"),
    (NULL, 3, 24.99, "01-May-2006");
INSERT INTO books VALUES
    ("0-672-31697-8", "Толстой", "Война и мир", 34.99),
    ("0-672-31745-1", "Булгаков", "Мастер и Маргарита", 24.99),
    ("0-672-31509-2", "Лермонтов", "Избранные произведения", 24.99),
    ("0-672-31769-9", "Пушкин", "Евгений Онегин", 49.99);
INSERT INTO order_items VALUES
    (1, "0-672-31697-8", 2),
    (2, "0-672-31769-9", 1),
    (3, "0-672-31769-9", 1),
    (3, "0-672-31509-2", 1),
    (4, "0-672-31745-1", 3);
INSERT INTO book_reviews VALUES
    ("0-672-31697-8", "Это замечательная книга, ее должен каждый прочитать.");
```

5.3 КОНТРОЛЬНЫЕ ВОПРОСЫ

1 В чем преимущество использования баз данных для хранения данных?

– базы данных сами заботятся о безопасности информации и ее сортировке;

– позволяют извлекать и размещать информацию при помощи одной строки;

– выборка информации из базы данных происходит значительно быстрее, чем из файла;

– код с использованием базы данных получается более компактным;

– обработка баз данных сводиться к последовательной обработке.

2 К какому типу СУБД относится MySQL?

– настольная реляционная СУБД;

– сетевая реляционная СУБД;

– сетевая иерархическая СУБД;

– настольная иерархическая СУБД;

– объектно-ориентированная реляционная СУБД.

3 Что такое клиент данных?

– часть СУБД, которая занимается приемом запросов от пользователей и выводом результатов;

– часть СУБД, с которой работает клиент;

– часть СУБД, которая выполняет только запросы на выборку и изменение;

– часть СУБД, которая непосредственно занимается взаимодействием с базой данных и собственно обработкой данных;

– часть СУБД, которая только выводит пользователю результаты запросов.

4 Что такое процессор данных?

– часть СУБД, которая занимается приемом запросов от пользователей и выводом результатов;

– часть СУБД, с которой работает клиент;

– часть СУБД, которая непосредственно занимается взаимодействием с базой данных и собственно обработкой данных;

– часть СУБД, которая выполняет только запросы на выборку и изменение;

– часть СУБД, которая только выводит пользователю результаты запросов.

5 Какие утверждения справедливы для серверных СУБД?

– нетребовательность к дополнительному программному обеспечению;

– невысокое быстродействие при многопользовательском доступе к базе данных по сети;

– большая производительность;

– недостаточная надежность и защищенность;

– сложность установки, настройки и сопровождения.

6 Какие утверждения справедливы для настольных СУБД?

– большая производительность;

– нетребовательность к дополнительному программному обеспечению;

– невысокое быстродействие при многопользовательском доступе к базе данных по сети;

– большая надежность и защищенность;

– сложность установки, настройки и сопровождения.

7 Какие утверждения справедливы для СУБД MySQL?

– MySQL – это программа-сервер;

– MySQL обрабатывает запросы и запоминает результат и по специальному запросу передает результат или его часть клиентским программам;

– MySQL – это клиентская программа;

– один сервер MySQL может поддерживать сразу несколько баз данных;

– один сервер MySQL может поддерживать только одну базу данных.

8 Какие типы данных поддерживает СУБД MySQL?

– ARRAY;
– BIGINT;
– DOUBLE;
– ENUM;
– BLOB.

9 Какие атрибуты могут быть заданы столбцу вместе с типом данных?

– AUTO_INCREMENT;
– NOT NULL;
– DEFAULT;
– PRIMARY KEY;
– TYPE.

10 Какая команда MySQL позволяет посмотреть список существующих баз данных?

– SHOW DATABASES;
– SHOW TABLES;
– SHOW db_name;
– SHOW DATABASES db_name;
– нет верных ответов.

11 Какая команда MySQL позволяет просмотреть список существующих в базе данных таблиц?

– SHOW DATABASES;
– SHOW TABLES;
– SHOW db_name;
– SHOW DATABASES db_name;
– нет верных ответов.

12 Какая команда MySQL выбирает активную базу данных?

– USE db_name;
– USE ;
– USE DATABASE db_name;
– USE TABLES db_name;
– нет верных ответов.

13 Какая команда MySQL предназначена для создания базы данных?

– CREATE DATABASE db_name;
– CREATE db_name;
– CREATE DATABASES;
– CREATE TABLES;
– нет верных ответов.

14 Какая команда MySQL предназначена для создания таблиц в базе данных?

– CREATE TABLE table_name (col_name type);
– CREATE TABLES;
– CREATE table_name (col_name type);

– CREATE TABLES (col_name type);

– нет верных ответов.

15 Выберите правильные варианты написания команды создания таблицы в MySQL:

– CREATE TABLE tel (fio text, adreess text, tel text);

– CREATE TABLE tel (fio text NOT NULL, adreess text, tel text);

– CREATE TABLE tel (fio text, adreess text, tel text, PRIMARY KEY(fio));

– CREATE TABLE tel (adreess text, tel text, PRIMARY KEY(fio));

– CREATE TABLE tel (fio, adreess, tel).

16 Какая команда MySQL предназначена для удаления баз данных?

– DROP DATABASE db_name;

– DROP DATABASES;

– DROP DATABASE;

– DROP DATABASES db_name;

– DROP db_name.

17 Какая команда MySQL предназначена для удаления таблицы из базы данных?

– DROP TABLE table_name;

– DROP TABLE;

– DROP TABLE;

– DROP TABLE table_name;

– DROP table_name.

18 Выберите правильные варианты написания команды добавления столбца в таблицу в MySQL:

– ALTER TABLE ADD adress AFTER;

– ALTER TABLE ADD adress text FIRST;

– ALTER TABLE my_friends ADD adress text ADD fio text;

– ALTER TABLE my_friends ADD adress text FIRST;

– ALTER TABLE my_friends ADD adress text AFTER fio.

19 Что делает команда ALTER TABLE table_name CHANGE … в MySQL?

– изменяет свойства одного или нескольких столбцов и при необходимости переименовывает их;

– изменяет свойства одного столбца;

– переименовывает столбец в таблице;

– изменяет свойства базы данных;

– изменяет и переименовывает свойства строки таблицы.

20 Что делает команда ALTER TABLE table_name DROP… в MySQL?

– изменяет свойство одного или нескольких столбцов и при необходимости переименовывает его;

– переименовывает один или несколько столбцов;

– изменяет свойства столбца;

– удаляет столбец;

– добавляет новую запись в таблицу.

21 С помощью какой команды MySQL можно добавить новую запись в таблицу?

– INSERT INTO;
– DELETE FROM;
– CREATE;
– ALTER TABLE ADD;
– ALTER TABLE CHANGE.

22 Что делает команда DELETE FROM … в MySQL?

– удаляет все записи, для которых выполняется условное выражение;
– удаляет все записи из таблицы;
– вычисляет условное выражение;
– выбирает записи, соответствующие условному выражению;
– удаляет все столбцы из таблицы.

23 Для чего предназначена команда UPDATE в MySQL?

– обновляет все записи в таблице;
– обновляет все столбцы в таблице;
– обновляет записи, для которых выполняется условное выражение;
– обновляет все записи и столбцы в таблице.