Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Поволжский государственный университет телекоммуникаций и информатики»

Кафедра «Прикладная информатика»

«УТВЕРЖДАЮ»

Зав. кафедрой ПИ
профессор _____ Маслов О.Н.

« 31 » августа 2019 г.

ЛАБОРАТОРНЫЕ РАБОТЫ

ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

«ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ»

для специальности 090301

Обсуждено на заседании кафедры

« 30 » августа 2019 г.

протокол № 1

Самара

2019

Лабораторная работа 1

СУБД MySQL

Цель работы: получить навыки работы с СУБД MySQL.

Практическое задание выполняется в локальной сети на рабочей станции с операционной системой Windows 95/98, 2000, XP или более поздней, с установленным сервером с операционной системой Linux версии не ниже пятой.

Порядок выполнения лабораторного задания

Отчёт по работе должен содержать следующее:

- Текст задания.
- Перечень всех использованных в практической работе команд и инструкций.
- Вывод по работе.

Общие сведения

В СУБД MySQL реализовано подмножество языка SQL, соответствующее спецификации ANSI SQL 92.

MySQL – это система управления базами данных.

База данных представляет собой структурированную совокупность данных. Эти данные могут быть любыми - от простого списка предстоящих покупок до перечня экспонатов картинной галереи или огромного количества информации в корпоративной сети. Для записи, выборки и обработки данных, хранящихся в компьютерной базе данных, необходима система управления базой данных, каковой и является СУБД MySQL. Поскольку компьютеры замечательно справляются с обработкой больших объемов данных, управление

базами данных играет центральную роль в вычислениях. Реализовано такое управление может быть по-разному как в виде отдельных утилит, так и в виде кода, входящего в состав других приложений.

MySQL – это система управления реляционными базами данных.

В реляционной базе данных данные хранятся не все скопом, а в отдельных таблицах, благодаря чему достигается выигрыш в скорости и гибкости. Таблицы связываются между собой при помощи отношений, благодаря чему обеспечивается возможность объединять при выполнении запроса данные из нескольких таблиц. SQL как часть системы MySQL можно охарактеризовать как язык структурированных запросов плюс наиболее распространенный стандартный язык, используемый для доступа к базам данных.

Программное обеспечение MySQL - это программное обеспечение (ПО) с открытым кодом.

ПО с открытым кодом означает, что применять и модифицировать его может любой желающий. Такое ПО можно получать по Internet и использовать бесплатно. При этом каждый пользователь может изучить исходный код и изменить его в соответствии со своими потребностями. Использование программного обеспечения MySQL регламентируется лицензией GPL (GNU General Public License), http://www.gnu.org/licenses/, в которой указано, что можно и чего нельзя делать с этим программным обеспечением в различных ситуациях.

В каких случаях следует отдавать предпочтение СУБД MySQL?

MySQL является очень быстрым, надежным и легким в использовании. Если вам требуются именно эти качества, попробуйте поработать с данным сервером. MySQL обладает также рядом удобных возможностей, разработанных в тесном контакте с пользователями. Первоначально сервер MySQL разрабатывался для управления большими базами данных с целью обеспечить более высокую скорость работы по сравнению с существующими

на тот момент аналогами. И вот уже в течение нескольких лет данный сервер успешно используется в условиях промышленной эксплуатации с высокими требованиями. Несмотря на то что MySQL постоянно совершенствуется, он уже сегодня обеспечивает широкий спектр полезных функций. Благодаря своей доступности, скорости и безопасности MySQL очень хорошо подходит для доступа к базам данных по Internet.

Технические возможности СУБД MySQL

ПО MySQL является системой клиент-сервер, которая содержит многопоточный SQL-сервер, обеспечивающий поддержку различных вычислительных машин баз данных, а также несколько различных клиентских программ и библиотек, средства администрирования и широкий спектр программных интерфейсов (API).

Типы данных

Обработка числовых данных

В MySQL есть пять целых типов данных, каждый из которых может быть со знаком (по умолчанию) или без знака (при добавлении слова *UNSIGNED* после имени типа).

Имя типа	**Кол-во бит**	**Диапазон со знаком**	**Диапазон без знака**
TINYINT	8	-128..127	0..255
SMALLINT	16	-32768..32767	0..65535
MEDIUMINT	24	-8388608..8388607	0..16777215
INTEGER	32	-147483648..2147483647	0..4294967295
BIGINT	64	$-(2^{63})..(2^{63}-1)$	$0..(2^{64})$

Тип *INT* используется как псевдоним для *INTEGER*. Другими псевдонимами служат: *INT1=TINYINT, INT2=SMALLINT, INT3=MEDIUMINT, INT4=INT, INT8=BIGINT и MIDDLEINT=MEDIUMINT.*

Обработка строковых данных

MySQL поддерживает следующие строковые типы (*M* обозначает максимальный отображаемый размер или PRECISION).

CHAR(M) – *с*трока фиксированной длины всегда дополняемая справа пробелами. M может находиться в диапазоне от 1 до 255 символов.

VARCHAR(M) – *с*трока переменной длины. Замыкающие пробелы удаляются базой данных при записи значения, что отличается от спецификации ANSI SQL. M может находиться в диапазоне от 1 до 255 символов.

ENUM(' value1' , 'value2',...) – перечисление. Строковый объект, у которого может быть только одно значение, выбранное из заданного списка (или NULL). В перечислении может быть до 65 535 различных значений.

SET('value1' , ' value2' , ..) **–** множество. Строковый объект, у которого может быть несколько значений (или ни одного), каждое из которых должно выбираться из указанного списка. SET может иметь максимум 64 элемента.

Длина типов *CHAR* и *VARCHAR* ограничена 255 байтами. Все строковые типы поддерживают двоичные символы, включая *NULL* (для литеральных строк используйте метод *$dbh->quote()*).

Поддерживаются также следующие псевдонимы:

BINARY(num) CHAR(num) BINARY

CHAR VARYING VARCHAR

LONG VARBINARY BLOB

LONG VARCHAR TEXT

VARBINARY(num) VARCHAR(num) BINARY

Основные команды

Просмотреть список команд программы **mysql** можно, запустив ее с параметром **--help**:

shell> mysql --help

Выведенный этой командой на экран монитора список команд достаточно хорошо прокомментирован и поэтому здесь не приводится.

Запустить СУБД MySQL можно следующим образом:

mysql [OPTIONS] database

Описание СУБД MySQL

Клиентская часть СУБД MySQL названа **mysql**. Она обеспечивает интерфейс командной строки с СУБД MySQL и возможность неинтерактивной пакетной обработки.

Опции, поддерживаемые программой mysql, представлены в таблице. Вы можете использовать или «короткий» одиночный символ или более подробную версию.

Опции mysql Таблица 1

Опция	**Описание**
-\?, --help	Справка

Продолжение табл. 1

Опция	**Описание**
-d, --debug=[options]	Вывести в протокол отладочную информацию. В общем виде *'d:t:o,filename'*
-d, --debug-info	Вывести отладочную информацию при выходе из программы
-e, --exec	Выполнить команду и выйти, неявная форма опции *--batch*
-f, --force	Продолжить, даже если мы

	сталкиваемся с SQL ошибкой
-h, --hostname=[hostname]	Задает имя сервера, с которым вы желаете соединиться
-P, --port=[port]	Порт для соединения с сервером MySQL
-p, --password=[password]	Пароль пользователя для соединения с сервером MySQL. Обратите внимание, что не должно быть пробела между -p и паролем
-q, --quick	Быстрый (небуферизованный вывод), может замедлить сервер, если вывод приостановлен
-s, --silent	Работать молча (подавить вывод)
-u, --user=[user]	Имя пользователя для соединения с сервером MySQL. Необязательно, если имя пользователя такое же, как ваш *логин*. По умолчанию именно ваш *логин* используется в качестве имени пользователя, что облегчает настройку

Окончание табл. 1

Опция	**Описание**
-v, --verbose	Подробный вывод. -v опция может быть удвоена или утроена для более подробного вывода. В программах русских авторов обычно именуется "уровнем болтливости программы"
-w, --wait	Если подключение терпит неудачу, то подождать и повторить попытку
-B, --batch	Выполнить в пакетном режиме.

	Никаких запросов и никаких ошибок в STDOUT. Устанавливается автоматически при чтении из записи в канал (пайп). Результаты будут выведены в формате с разделением табуляцией. Одна строка результата соответствует одной строке вывода
-I, --help	Справка, эквивалент -\?
-V, --version	Вывести информацию о версии пакета

В интерактивном режиме mysql будет печатать результаты в таблице подобно примеру, приведенному ниже. Если не задан пароль или имя пользователя mysql попробует зайти в систему на сервере базы данных с использованием вашего логина и НУЛЕВОГО (ПУСТОГО) пароля. Если ваш mysql логин отличается от вашего логина в unix, или если вы имеете пароль, то соединение с MySQL выполнено не будет.

Например:

bsd# mysql -u coobic

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 4 to server version: 4.0.12-max

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

Реализация языка SQL в СУБД MySQL

Создание базы данных

Просмотреть список существующих баз данных в настоящее время на сервере можно при помощи команды **SHOW**:

mysql> **SHOW DATABASES;**

+----------+

| Database |

```
+----------+
| mysql    |
| test     |
| trf      |
+----------+
3 rows in set (0.00 sec)
```

База данных **mysql** просто необходима, так как в ней описываются пользовательские права доступа. База **test** часто применяется для экспериментов.

Впрочем, всех баз вы можете и не увидеть, если у вас нет привилегии SHOW DATABASES.

Если база данных **test** существует, попробуйте обратиться к ней:

```
mysql> USE test
Database changed
```

В команде **USE**, как и **QUIT**, точка с запятой не нужна (конечно, данные команды тоже можно завершать точкой с запятой – никакого вреда от этого не будет). Команда **USE** отличается от остальных и кое-чем еще: она должна задаваться одной строкой.

Если администратор при выдаче разрешения создаст для вас базу, с ней можно сразу начинать работу. При отсутствии БД вам придется создать ее самостоятельно:

```
mysql> CREATE DATABASE perpetuum;
Query OK, 1 row affected (0.00 sec)
```

В Unix имеет значение регистр символов в именах баз данных (в отличие от ключевых слов SQL), так что в этой ОС вам всегда придется называть свою базу **perpetuum**, а не Perpetuum или еще как-нибудь. Это же правило распространяется и на имена таблиц (в Windows данное ограничение не

действует, однако при обращении к базам и таблицам в пределах одного запроса, тем не менее, можно использовать только один регистр).

При создании базы данных она автоматически не выбирается; выбирать ее нужно отдельно. Сделать **perpetuum** текущей базой можно с помощью следующей команды:

```
mysql> USE perpetuum
Database changed
```

Создавать базу нужно только однажды, но выбирать ее приходится в каждом сеансе работы с **mysql**. Делать это можно с помощью команды USE, представленной выше. А можно выбирать базу и из командной строки при запуске **mysql**. Для этого достаточно лишь ввести ее имя после параметров соединения, которые нужно вводить в любом случае. Например:

```
shell> mysql -h host -u user -p perpetuum
Enter password: ********
```

Обратите внимание: в вышеприведенной команде perpetuum не является паролем. Ввести пароль в командной строке после параметра -p можно без пробела (например, **-pmypassword**, а не **-p mypassword**). Впрочем, пароль в командной строке все равно лучше не вводить, так как таким образом его могут и подсмотреть.

Создание таблицы

При помощи команды **CREATE TABLE** определим структуру новой таблицы:

```
mysql> CREATE TABLE TABLE1 (
    -> NUMBER INTEGER NOT NULL AUTO_INCREMENT,
```

```
    -> FAMALY VARCHAR(35),
    -> NAME VARCHAR(30),
    -> OTCHESTVO VARCHAR(35),
    -> DESCRIPTION BLOB,
    -> PHOTOFILE BLOB,
    -> EMAIL VARCHAR(40),
    -> ZVANIE VARCHAR(34),
    -> DOLGNOST VARCHAR(40),
    -> PRIMARY KEY (NUMBER)
    -> );
Query OK, 0 rows affected (0.06 sec)
```

Теперь, когда таблица создана, команда **SHOW TABLES** должна вывести следующее:

```
mysql> SHOW TABLES;
+--------------------+
| Tables_in_perpetuum |
+--------------------+
| TABLE1             |
+--------------------+
1 row in set (0.00 sec)
```

Проверить, правильно была ли таблица создана в соответствии с планом, можно при помощи команды **DESCRIBE:**

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
```

```
| name    | varchar(20) | YES |     | NULL    |       |
| owner   | varchar(20) | YES |     | NULL    |       |
| species | varchar(20) | YES |     | NULL    |       |
| sex     | char(1)     | YES |     | NULL    |       |
| birth   | date        | YES |     | NULL    |       |
| death   | date        | YES |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

Использовать команду **DESCRIBE** можно в любое время, например, если вы забудете имена столбцов или типы, к которым они относятся.

mysql> **describe TABLE1;**

```
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| NUMBER      | int(11)     |      | PRI | NULL    | auto_increment |
| FAMALY      | varchar(35) | YES  |     | NULL    |                |
| NAME        | varchar(30) | YES  |     | NULL    |                |
| OTCHESTVO   | varchar(35) | YES  |     | NULL    |                |
| DESCRIPTION | blob        | YES  |     | NULL    |                |
| PHOTOFILE   | blob        | YES  |     | NULL    |                |
| EMAIL       | varchar(40) | YES  |     | NULL    |                |
| ZVANIE      | varchar(34) | YES  |     | NULL    |                |
| DOLGNOST    | varchar(40) | YES  |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
```

9 rows in set (0.02 sec)

Загрузка данных в таблицу

Создав таблицу, нужно позаботиться о ее заполнении. Для этого предназначены команды **LOAD DATA** и **INSERT**.

mysql> **INSERT INTO TABLE1 VALUES (NULL,**
-> **'Kulikov', 'Denis', 'Alexandrovich',**
-> **'coobic', NULL, 'denis@redcom.ru'**,
-> **NULL, NULL);**

Query OK, 1 row affected (0.03 sec)

Команда SELECT

Синтаксис:

```
SELECT [STRAIGHT_JOIN] [DISTINCT | ALL] select_expression,...
  [FROM tables... [WHERE where_definition] [GROUP BY column,...]
  [ORDER BY column [ASC | DESC], ...] HAVING full_where_definition
  [LIMIT [offset,] rows] [PROCEDURE procedure_name]]
  [INTO OUTFILE 'file_name'... ]
```

Здесь where_definition:

```
 where_definition:
    where_expr or where_expr [AND | OR] where_expr
```

where_expr имеет формат:

```
 where_expr:
    column_name [> | >= | = | <> | <= | <]
    column_name_or_constant or column_name LIKE column_name_or_constant or
    column_name IS NULL or column_name IS NOT NULL or (where_definition)
```

Описание команды SELECT

Оператор **SELECT** является краеугольным камнем всего языка SQL. Он используется, чтобы выполнить запросы к базе данных. В MySQL версии меньше 3.21.x использование предложения **WHERE** очень ограничено. **HAVING** будет работать там, где предложение **WHERE** ничего не делает. В основном, использовать функции с **WHERE** нельзя, но можно использовать функции с **HAVING. HAVING** по существу есть **WHERE** применительно к результатам. Он используется главным образом для узкой области данных, возвращенных запросом. Вы должны иметь права ***select*** для использования **SELECT**.

Ниже приведена простая команда, запрашивающая у сервера информацию о его версии и текущей дате. Введите ее в командной строке mysql> и нажмите Enter:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+------------+--------------+
| version()  | current_date |
+------------+--------------+
| 4.0.12-max | 2003-10-08   |
+------------+--------------+
1 row in set (0.00 sec)
mysql>
```

Этот запрос иллюстрирует следующие особенности mysql:

- Команда обычно состоит из SQL-выражения, за которым следует точка с запятой. Из этого правила есть и исключения – команды без точки с запятой. Одним из них является упомянутая выше команда QUIT, остальные мы рассмотрим позднее.
- Когда пользователь вводит команду, **mysql** отправляет ее серверу для выполнения и выводит на экран сначала результаты, а затем - новую строку **mysql>,** что означает готовность к выполнению новых команд.
- **mysql** выводит результаты работы запроса в виде таблицы (строк и столбцов). В первой строке этой таблицы содержатся заголовки столбцов, а в следующих строках - собственно результаты. Обычно заголовками столбцов становятся имена, полученные из таблиц базы. Если же извлекается не столбец таблицы, а значение выражения (как это происходит в приведенном выше примере), **mysql** дает столбцу имя запрашиваемого выражения.
- **mysql** сообщает количество возвращаемых строк и время выполнения запроса, что позволяет в некоторой степени составить представление о производительности сервера. Эти значения обычно

весьма впечатляют, так как представляют обычное (а не машинное время), кроме того, на них оказывает влияние загрузка сервера и скорость работы сети (для сокращения размеров листингов в остальных примерах этой главы строка "rows in set" удалена).

Для ввода ключевых слов можно использовать любой регистр символов. Приведенные ниже запросы абсолютно идентичны:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

А это – еще один запрос. В нем демонстрируется использование **mysql** в качестве несложного калькулятора:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+------------+---------+
| SIN(PI()/4) | (4+1)*5 |
+------------+---------+
|   0.707107 |      25 |
+------------+---------+
```

Все команды, представленные выше, были относительно короткими и состояли из одной строки. В одну строку можно поместить и несколько команд. Но каждая из них должна заканчиваться точкой с запятой:

```
mysql> SELECT VERSION(); SELECT NOW();
+------------+
| VERSION()  |
+------------+
```

```
| 4.0.12-max |
+------------+
1 row in set (0.00 sec)

+---------------------+
| NOW()               |
+---------------------+
| 2003-10-08 23:22:38 |
+---------------------+
1 row in set (0.00 sec)
```

Вот пример несложного выражения, занимающего несколько строк:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+-----------------+--------------+
| user()          | current_date |
+-----------------+--------------+
| coobic@localhost | 2003-10-08   |
+-----------------+--------------+
1 row in set (0.00 sec)
```

Если вы решите отменить исполнение набираемой команды, наберите **\c:**

```
mysql> SELECT
    -> USER()
    -> \c
```

mysql>

Обратите внимание на метку: после ввода команды **\c** она снова принимает вид mysql>, показывая, что программа **mysql** перешла в режим ожидания указаний.

В таблице 2 приведены все возможные варианты вида метки командной строки и соответствующие им состояния **mysql:**

Таблица 2

Метка	**Значение**
mysql>	Ожидание новой команды
->	Ожидание следующей строки многострочной команды
'>	Ожидание следующей строки, сбор строкового выражения, начинающегося с одиночной кавычки (''')
">	Ожидание следующей строки, сбор строкового выражения, начинающегося с двойной кавычки (""')

Обычно многострочные команды получаются случайно, когда хочешь создать обычную команду, но забываешь поставить завершающую точку с запятой. В таком случае **mysql** ожидает продолжения:

mysql> **SELECT USER()**
    ->

Если с вами произошло подобное (вы думаете, что завершили команду, но программа выдает только метку ->), то **mysql**, вероятнее всего, ждет точки с запятой. Не обратив внимание на метку командной строки, можно довольно долго ждать выполнения команды, не понимая в чем дело. А достаточно лишь поставить точку с запятой, завершив команду, которую **mysql** и выполнит:

```
mysql> SELECT USER()
    -> ;
+-----------------+
| user()          |
+-----------------+
| coobic@localhost |
+-----------------+
1 row in set (0.00 sec)
```

Метки '> и "> используются при сборе строк. В MySQL строки можно заключать как в одинарные ('), так и в двойные (") кавычки (можно, например, написать **'hello'** или **"goodbye"),** к тому же, mysql позволяет вводить строковые выражения, состоящие из нескольких строчек текста. Метка '> или "> обозначает, что вы ввели строку, открывающуюся символом кавычек ' или ", но еще не ввели завершающую строковое выражение закрывающую кавычку.

Это, конечно, нормально, если вы собираетесь создать большое строковое выражение из нескольких строчек. Но это не слишком частый случай. Гораздо чаще оказывается, что вы просто забыли поставить закрывающую кавычку. Например:

```
mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;
    ">
```

Если ввести такую команду **SELECT**, нажать Enter и подождать результатов, ничего не произойдет. Тут-то и нужно обратить внимание на метку командной строки, выглядящую вот так: **">.** Это значит, что **mysql** ждет ввода завершающей части строки. (Теперь заметили ошибку в команде? В строке **"Smith** нет закрывающей кавычки.)

Что делать в этом случае? Проще всего было бы отменить команду. Однако теперь просто набрать **\c** нельзя, так как mysql примет эти символы за часть собираемой строки! Вместо этого нужно ввести закрывающие кавычки (тем самым дав mysql понять, что строка закончилась) и лишь затем набрать \c:

mysql> **SELECT * FROM my_table WHERE name = "Smith AND age < 30;**

"> "\c

mysql>

Метка командной строки снова примет вид mysql>, показывая готовность **mysql** к выполнению команд.

Знать значение меток '> и "> необходимо, так как при вводе незавершенной строки все последующие строки будут игнорироваться mysql – включая строку с командой **QUIT**! Это может основательно сбить с толку, особенно, если не знать, что для отмены команды перед соответствующей последовательностью символов необходимо поставить закрывающую кавычку.

Команда Delete

Синтаксис команды DELETE полностью соответствует спецификации SQL92, поэтому в данных методических указаниях не приводится.

Особенности СУБД

Ядро MySQL само удаляет пробелы, находящиеся в конце строки. Другая характерная особенность состоит в том, что при сравнениях и сортировке регистр символов в колонках *CHAR* и *VARCHAR* не учитывается, если только не задать атрибут *BINARY,* например:

CREATE TABLE foo (A VARCHAR(10)BINARY);

В версиях MySQL после 3.23 можно осуществлять сравнение строк без учета регистра с помощью модификатора BINARY:

*SELECT * FROM table WHERE BINARY column = "A"*

Национальные символы обрабатываются в сравнениях соответственно системе кодировки, указанной во время компиляции, по умолчанию ISO-8859-1. Системы кодировки, не соответствующие ISO, в частности UTF-16, не поддерживаются.

Конкатенация строк производится с помощью функции SQL CONCAT(s1, s2, ...).

Обработка данных типа «дата»

MySQL поддерживает следующие типы даты и времени:

DATE

Дата в диапазоне с 0000-01-01 по 9999-12-31. MySQL выводит значения DATE в формате YYYY-MM-DD, но позволяет присваивать значения колонкам DATE в таких форматах:

YYMMDD

YYYYMMDD

YY..MM.DD

YYYY.MM.DD

Здесь «.» может быть любым нецифровым разделителем, а при задании года двумя цифрами предполагается год 20YY, если YY меньше 70.

DATETIME

Комбинация даты и времени. Поддерживаемый диапазон от 0000-01-01 00:00:00 до 9999-12-31 23:59: 59. MySQL выводит значения DATE-TIME в формате YYYY-MM-DD HH:MM:SS, но допускает присвоение значений колонкам DATETIME с использованием показанных выше форматов для DATE с добавлением HH: MM: SS.

TIMESTAMP(M)

Временная метка. Диапазон от 1970-01-01 00:00:00 и до некоторого момента в 2032 или 2106 году, в зависимости от специфического для операционной системы типа time_t. MySQL выводит значения TIMESTAMP в

форматах *YYYYMMDDHHMMSS*, *YYMMDDHHMMSS*, *YYYYMMDD* или *YYMMDD*, в соответствии со значением M, равным 14 (или же отсутствием подобного значения), 12, 8 или 6, но присваивать значения колонкам TIMESTAMP позволяет с помощью строк или чисел. Этот формат вывода не согласуется с руководством, поэтому его нужно проверять в каждой версии, т. к. он может измениться.

Колонки типа TIMESTAMP удобно использовать для регистрации времени операций INSERT или UPDATE, поскольку если для них не указывается значение, им присваивается время последней операции. Текущее время устанавливается и при попытке присвоить значение *NULL*.

TIME

Время в диапазоне от-838:59:59 до 838:59:59. MySQL выводит значения времени в формате HH:MM:SS. Присваивать значения колонкам TIME можно с помощью формата *[[[DAYS] [H]H:*

]MM:]SS[:fraction] или *[[[[[H]H]H]H]MM]SS [.fraction]*.

YEAR

Год. Допустимыми значениями являются 1901-2155 и 0000 в формате четырех знаков и 1970-2069 при использовании двух цифр (70-69). При вводе двузначные годы в диапазоне 00-69 воспринимаются как 2000-2069. (YEAR является новым типом данных в MySQL 3.22.)

Обработка данных типа LONG/BLOB

В MySQL поддерживаются следующие типы BLOB:

TINYBLOB / TINYTEXT максимальный размер 255 (2**8 - 1)
BLOB / TEXT максимальный размер 65535 (2**16 - 1)
MEDIUMBLOB / MEDIUMTEXT максимальный размер 16777215 (2**24 - 1)
LONGBLOB / LONGTEXT максимальный размер 4294967295 (2**32 - 1)

Во всех типах BLOB допускаются двоичные символы.

Максимальный размер значений параметров метода *bind_param()* ограничен только максимальным размером команды SQL. По умолчанию это 1

Мбайт, но можно его увеличить почти до 24 Мбайт, изменив в *mysqld* переменную *max_allowed_packet*.

Передавать TYPE или другие атрибуты методу *bind_param()* при связывании этих типов не нужно.

Имена таблиц и колонок

В MySQL имена таблиц и колонок должны быть не длиннее 64 символов.

В MySQL можно заключать имена таблиц и колонок в одиночные кавычки (использование стандартных двойных кавычек допустимо, если база данных запущена с ключом --ansi-mode).

Это необходимо делать, если имя содержит специальные символы или зарезервированное слово. Ограничения на имена таблиц вызваны тем, что таблицы хранятся в файлах и имена таблиц являются, в действительности, именами файлов. В частности, чувствительность имен таблиц к регистру определяется файловой системой, также недопустимы некоторые символы, например «.» и «/».

В именах колонок регистр не различается, но имена сохраняются без преобразования регистра.

В MySQL в именах могут использоваться символы национальных алфавитов (с установленным восьмым битом).

Автоматическая генерация ключей и последовательностей

Для всех целочисленных полей в MySQL можно установить атрибут *AUTO_INCREMENT*. Это значит, что если есть таблица:

CREATE TABLE a (

id INTEGER AUTO_INCREMENT NOT NULL PRIMARY KEY,

...)

и команда:

INSERT INTO a (id,...) VALUES (NULL...),

то автоматически генерируется уникальный ID (то же, если поле ID вообще не указывать в команде вставки). Сгенерированный ID позднее можно извлечь:

*$sth->**{mysql_insertid} (1 21_xx)***

*$sth->**{insertid} (1 20_xx)***

Либо, если вы используете *$dbh->do*, а не *prepare/execute*, выполните команды:

*$dbh->**{mysql_insertid} (1 21_xx)***

*$dbh->**do("SELECT LAST_INSERT_ID()"); (1 20_xx)***

MySQL не поддерживает непосредственно генераторы последовательностей, но они легко могут эмулироваться (детали можно найти в руководстве по MySQL). Например:

UPDATE seq SET id=last_insert_id(id+1)

Задание на лабораторную работу

1. Ознакомиться с операциями, производимыми в консоли СУБД и выполнить следующие действия:
    - создать новую удаленную базу данных MySQL на сервере, названием которой будет is_№_в_журнале, пароль для доступа №_в_журнале;
    - просмотреть доступные пользователю базы данных и версию СУБД;
    - создать в созданной базе данных одну таблицу;
    - отобразить структуру таблицы;
    - добавить 5 записей в таблицу;
    - просмотреть все записи в таблице;
    - удалить одну запись в таблице;

- очистить таблицу;
- проверить различные варианты выборки (поиска) по значению одного из полей таблицы.

2. Таблица должна содержать минимум 5 строковых полей, 2 – цифровых, 1 – для хранения двоичных файлов (рисунков, фотографий).
3. Убедиться в работоспособности СУБД и продемонстрировать ее работу преподавателю.
4. Составить отчет по лабораторной работе.

Содержание отчета

1. Цель работы.
2. SQL-скрипт, использованный для создания БД.
3. Структура созданной таблицы.
4. Пример наполнения таблицы.
5. Версия СУБД и доступные пользователю базы данных.

Контрольные вопросы

1. Основные возможности СУБД MySQL и поддерживаемые платформы.
2. Типы данных, с которыми работает СУБД MySQL.
3. Команды, используемые для подключения к СУБД в консольном режиме.
4. Правила работы в консольном режиме с СУБД MySQL.
5. Особенности работы с СУБД MySQL.

Библиографический список

1. Документация к серверу MySQL. – Электр. дан. – Режим доступа: http://www.mysql.org

2. Чекалов А. Прагматический подход к разработке приложений Web баз данных. – Электр. дан. – Режим доступа: Web-сервер Citforum http://www.citforum.ru/internet/webdbapp/ index.shtml

Лабораторная работа 2

Создание баз данных и таблиц в среде MYSQL. Информационное наполнение

Цель работы: Ознакомиться с возможностями СУБД MySQL и создать с его помощью базу данных, набор таблиц в ней и заполнить таблицы данными для последующей работы.

Содержание работы и методические указания к ее выполнению

1. Ознакомиться с возможностями работы клиентского приложения MySQL .
2. Изучить набор команд языка SQL, связанный с созданием базы данных, созданием, модификацией структуры таблиц и их удалением, вставкой, модификацией и удалением записей таблиц.

Функция	Описание
CREATE DATABASE DB_NAME	создание базы данных
USE DATABASE	выбор существующей базы данных
CLOSE DATABASE	закрытие файлов текущей базы данных
DROP DA TABASE	удаление базы данных
CREATE TABLE	создание таблицы базы данных
ALTER TABLE	модификация структуры базы данных
DROP TABLE	удаление таблицы базы данных
INSERT	добавление одной или нескольких строк в таблицу
DELETE	удаление одной или нескольких строк из таблицы
UPDATE	модификация одной или нескольких строк таблицы
LOAD DATA INFILE	загрузка данных в таблицы из файла

3. Создать базу данных.

Создание базы данных в MySQL производится с помощью утилиты mysqladmin. Изначально существует только БД mysql для администратора и БД test, в которую может войти любой пользователь и которая по умолчанию пуста. Приведенный ниже пример иллюстрирует создание базы данных.

```
Mysql/bin>mysqladmin -u root -p create data_name
Enter password:******
Database "data_name" created.

mysqlbin>
```

Где data_name – имя создаваемой БД. Проверить, что БД создана можно ранее рассмотренной командой ***Show databases*** или утилитой **mysqlshow.**

По умолчанию, **root** имеет доступ ко всем базам данных и таблицам. Перейти в созданную базу данных можно, используя команду **mysql.** Use database

```
Mysql/bin>mysql -u root -p data1
Enter password:******

    Welcome to MySQL monitor.
```

Или, находясь в другой базе данных, например в mysql ввести команду:

```
mysql>use data1

Database changed.
```

Создать базу данных можно непосредственно находясь в клиентском приложении MySQL, вводом команды:

CREATE DATABASE Base_name

Где **Base_name** имя создаваемой базы данных. В созданной базе можно создавать таблицы и вводить информацию. Указанные операции можно выполнить, используя специализированное программное обеспечение, например MySQL-Front, запуск которого осуществляется из меню ПУСК/ПРОГРАММЫ (см. рис 2.1, 2.2).

Необходимо указать:

- Имя;
- Хост;
- Пароль;
- Порт;
- Имя БД (при необходимости).

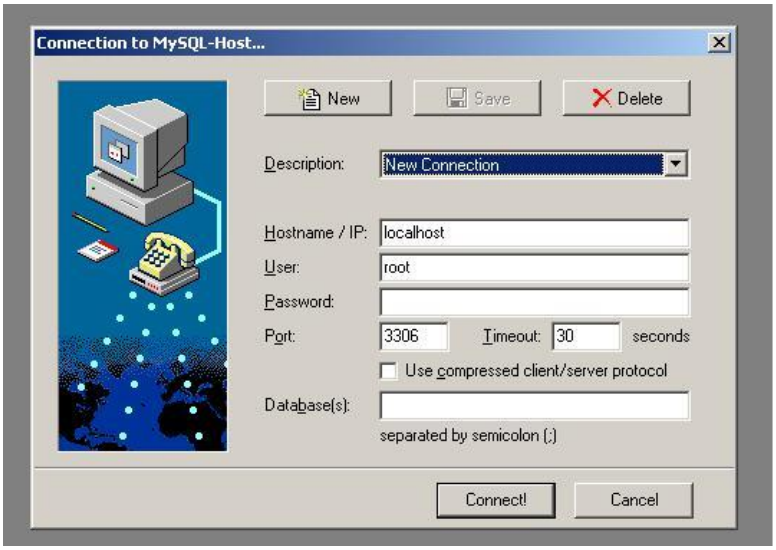

Рисунок 2.1 - Запуск MySQL-front

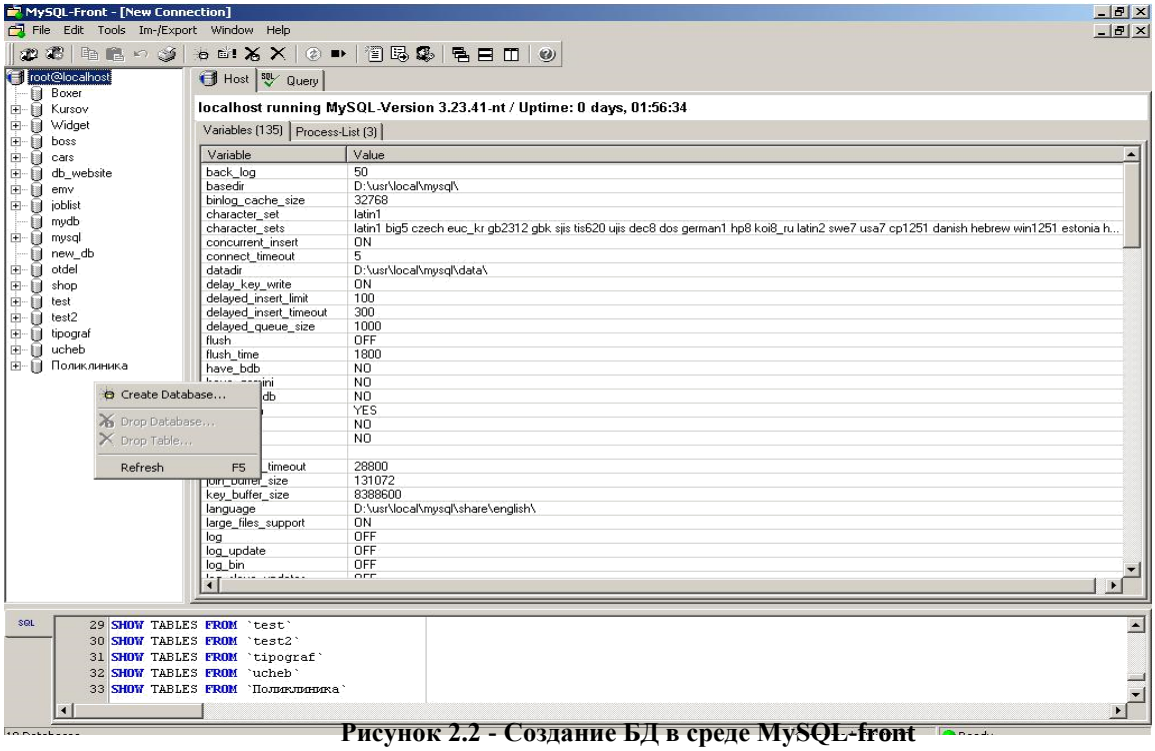

Рисунок 2.2 - Создание БД в среде MySQL-front

После задания активной БД можно с помощью средств, предоставляемых программой изменять структуру БД, вводить данные, задавать ключевые поля. Помимо этого можно в специально отведенном окне напрямую вводить инструкции, используя синтаксис языка SQL (рис. 2.3), как показано на рисунке:

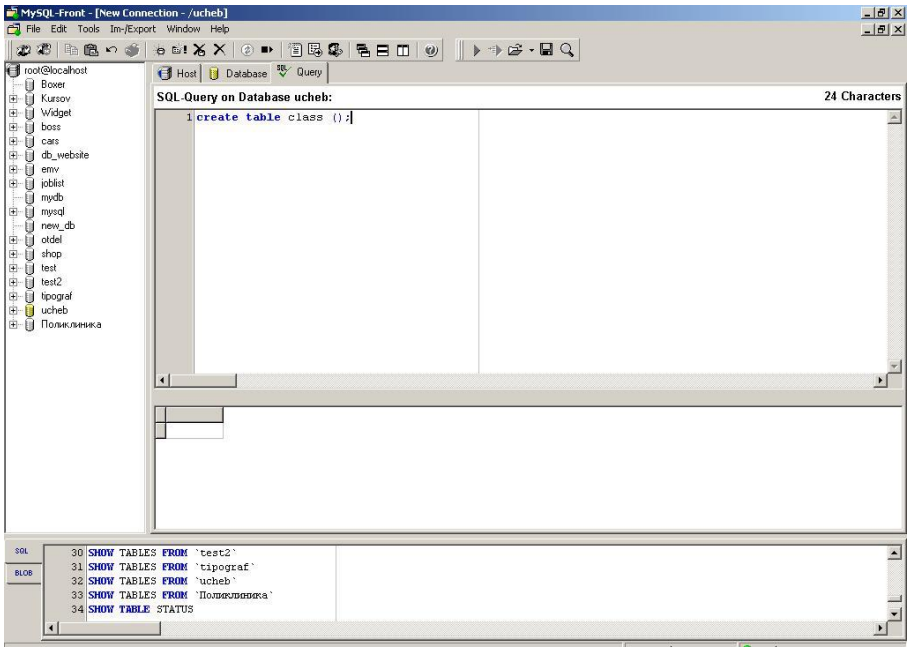

Рисунок 2.3 - Использование синтаксиса SQL

4. Средствами языка SQL необходимо создать четыре таблицы в базе данных, используя команду

CREATE TABLE, синтаксис которой приведен в приложении. Для таблицы J:

CREATE TABLE j (

```
    Jnum varchar(6) NOT NULL default '',
    Jnam varchar(20) default NULL,
    Ci varchar(20) default NULL,
    PRIMARY KEY (Jnum)
  ) TYPE=MyISAM;
```

Значками /* */ - выделяются комментарии в тексте запроса.

При создании таблиц выполнить такую реализацию, чтобы она отражала структуру таблиц, указанную ниже (таблице S, P, J, SPJ) и должны быть наложены следующие ограничения:

- поля номер_поставщика, номер_детали, номер_изделия во всех таблицах имеет символьный тип и длину 6 (**varchar**(6));
- поля рейтинг, вес и количество имеют целочисленный тип (**integer)**;
- поля фамилия, город (поставщика, детали или изделия), название (детали или изделия) имеют символьный тип и длину 20 (**varchar**(20));
- ни для одного поля не предусматривается использование индексов;
-для всех полей допускаются значения NULL и значения-дубликаты, кроме полей первичного и внешнего ключей.

После создания пустых таблиц их необходимо наполнить данными. Вводить данные в нее можно несколькими способами:

а)Вручную, используя команду **insert into**; Пример

ввода данных вручную (команда INSERT):

```
mysql>insert into J (Jnum, Jnam, Ci)values ('J1','Жесткий
диск','Париж'); или
mysql>insert into J values ('J1','Жесткий диск','Париж');
```

//т.е в случае если вы вставляете данные во все поля таблицы то их перечислять не обязательно.

Таким образом SQL инструкция имеет следующий вид

INSERT INTO table_name (id, name) VALUES ('id_value', 'name_value');

Записать и выполнить совокупность запросов для занесения нижеприведенных данных в созданные таблицы

insert into имя_таблицы [(поле [,поле]...)] values (константа [,константа]...)

б)Загрузить данные из текстового файла, что является более предпочтительным, особенно если нужно ввести несколько тысяч записей.

Синтаксис команды LOAD DATA INFILE.

```
DATA [LOW_PRIORITY] [LOCAL] INFILE 'file_name.txt' [REPLACE | IGNORE]
INTO TABLE tbl_name
[FIELDS
[TERMINATED BY 't']
```

[OPTIONALLY] ENCLOSED BY '']
[ESCAPED BY '']]
[LINES TERMINATED BY 'n']
[IGNORE number LINES]
[(col_name,...)]

Пример:

LOAD DATA LOCAL INFILE '/MyDocs/categories.txt' REPLACE INTO TABLE category FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '\"' LINES TERMINATED BY '\n'

В данном случае файл ***categories.txt*** находится на машине под управлением MS Windows, в каталоге ***C:\MyDocs***.

Обратите внимание на UNIX стиль написания пути.

Слово **REPLACE**
В SQL запросе означает, что необходимо замещать записи с совпадающими значениями ключей.

INTO TABLE[6]
указывает имя таблицы, куда будут импортированы данные.

FIELDS TERMINATED BY ';' [7]
указывает разделители полей, порядок полей должен быть таким же, как и в таблице назначения,

OPTIONALLY ENCLOSED BY '\"'
указывает, что поля VARCHAR взяты в двойные кавычки,

и LINES TERMINATED BY '\r' [8]

в) Использовать утилиту mysqlimport также для загрузки данных из текстового файла.

Эти и другие операции можно выполнить также и в программе MySQL-Front (рис. 2.4).

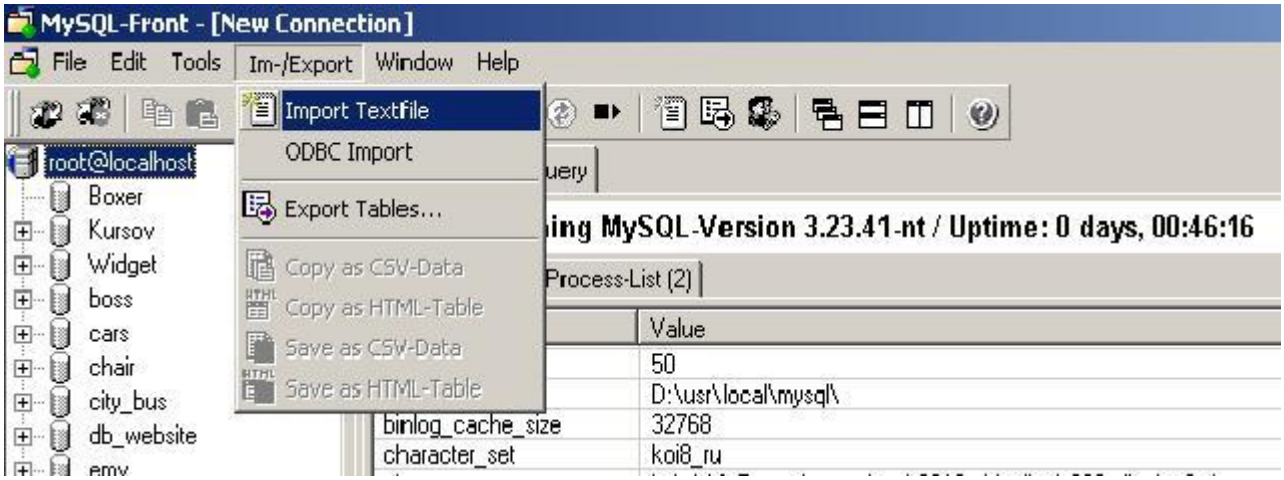

Рисунок 2.4 - Использование программы MySQL-front для заполнения таблиц данными из файла

6 В случае если данные вставляются не во все ячейки таблицы то это указывается при формировании инструкции tablename(id, id2), где tablename – имя таблицы, а id, id2 наименования атрибутов таблицы.

7 В случае использования табулятора \t

8 В случае Enter \r\n

Таблица поставщиков (S)

Номер поставщика	Фамилия	Рейтинг	Город
S1	Смит	20	Лондон
S2	Джонс	10	Париж
S3	Блейк	30	Париж
S4	Кларк	20	Лондон
S5	Адамс	30	Афины

Таблица деталей (P)

Номер детали	Название	Цвет	Вес	Город
P1	Гайка	Красный	12	Лондон
P2	Болт	Зеленый	17	Париж
P3	Винт	Голубой	17	Рим
P4	Винт	Красный	14	Лондон
P5	Кулачок	Голубой	12	Париж
P6	Блюм	Красный	19	Лондон

Таблица изделий (J)

Номер изделия	Название	Город
J1	Жесткий диск	Париж
J2	Перфоратор	Рим
J3	Считыватель	Афины
J4	Принтер	Афины
J5	Флоппи-диск	Лондон
J6	Терминал	Осло
J7	Лента	Лондон

Таблица поставок (SPJ)

Номер поставщика	Номер детали	Номер изделия	Количество
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J3	200

Номер поставщика	Номер детали	Номер изделия	Количество
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400
S2	P3	J7	800
S2	P5	J2	100
S3	P3	J1	200
S3	P4	J2	500
S4	P6	J3	300
S4	P6	J7	300
S5	P2	J2	200
S5	P2	J4	100
S5	P5	J5	500
S5	P5	J7	100
S5	P6	J2	200
S5	P1	J4	100
S5	P3	J4	200
S5	P4	J4	800
S5	P5	J4	400
S5	P6	J4	500

Убедиться в успешности выполненных действий. При необходимости исправить ошибки. Для ускорения процесса ввода данных рекомендуется воспользоваться командой LOAD DATA (синтаксис см. в приложении), предварительно скопировав содержимое перечисленных таблиц сначала в Excel, а оттуда в текстовые файлы. Такой порядок необходим, для того, чтобы текстовый файл был с табуляцией.

5. Выполнить модификацию структуры таблицы SPJ, добавив в SPJ поле с датой поставки. Убедиться в успешности выполненных действий. При необходимости исправить ошибки (команда Alter table).

6. Уничтожить созданные таблицы, предварительно сохранив инструкции для восстановления структуры БД и информационного наполнения, используя средства работы СУБД[9]. Убедиться в успешности выполненных действий.

7. Выполнить необходимые действия, написав и выполнив соответствующие запросы для модификации таблиц, чтобы структура соответствовала концеп-

[9] См утилиту Mysqldump

туальной модели учебной базы данных (рисунок 2.5). Убедиться в успешности выполненных действий. При необходимости исправить ошибки.

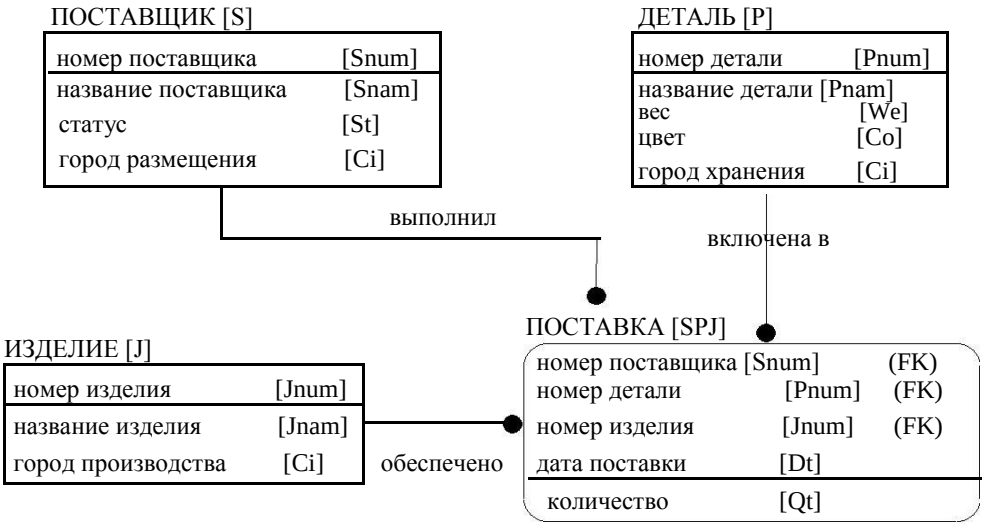

Рисунок 2.5 - Концептуальная модель учебной базы данных

Проверить результат заполнения таблиц, написав и выполнив простейший запрос:

select * from имя_таблицы

При наличии ошибок выполнить корректировку, исправив либо удалив ошибочные строки таблиц

Контрольные вопросы

1. В каких режимах возможно создание базы данных?
2. Какие типы данных допустимы при создании таблицы?
3. Как выполнить создание таблицы средствами СУБД?
4. Как выполнить создание таблицы средствами языка SQL?
5. Как разделяются операторы SQL в случае нескольких операторов в запросе?
6. Каким образом выполнить простейшие операции вставки строк данных в таблицу средствами SQL?
7. Каким образом выполнить простейшие операции модификации строк таблицы средствами SQL?
8. Каким образом выполнить просмотр таблицы?
9. Как получить информацию о структуре таблицы в рамках СУБД MySQL?

Лабораторная работа 3

Создание запросов и модификация таблиц базы данных

Цель работы: Используя данные базы данных, подготовленной в предыдущей лабораторной работе, подготовить и реализовать серию запросов, связанных с выборкой информации и модификацией данных таблиц.

Содержание работы и методические указания к ее выполнению

1. Изучить набор команд языка SQL, связанный с созданием запросов, добавлением, модификацией и удалением строк таблицы:
select - осуществление запроса по выборке информации из таблиц базы данных;
insert - добавление одной или нескольких строк в таблицу;
delete - удаление одной или нескольких строк из таблицы;
update - модификация одной или нескольких строк
таблицы; **union** - объединение запросов в один запрос.
2. Изучить состав, правила и порядок использования ключевых фраз оператора select:
select - описание состава данных, которые следует выбрать по запросу (обязательная фраза);
from - описание таблиц, из которых следует выбирать данные (обязательная фраза);
where - описание условий поиска и соединения данных при запросе;
group by - создание одной строки результата для каждой группы (группой называется множество строк, имеющих одинаковые значения в указанных столбцах);
having - наложение одного или более условий на группу;
order by - сортировка результата выполнения запроса по одному или нескольким столбцам;
into outfile - создание файла, в который будет осуществлен вывод результатов соответствующего запроса.
Порядок следования фраз в команде select должен соответствовать приведенной выше последовательности. Для лучшего понимания механизма функционирования выполните следующие упражнения:

I. Простые запросы на языке SQL

Запрос на языке SQL формируется с использованием оператора Select. Оператор Select используется

- для выборки данных из базы данных;
- для получения новых строк в составе оператора Insert;
- для обновления информации в составе оператора Update.

В общем случае оператор Select содержит следующие семь спецификаторов, расположенных в операторе в следующем порядке:

- спецификатор Select;
- спецификатор From;
- спецификатор Where;
- спецификатор Group by;
- спецификатор Having;
- спецификатор Order by;

Обязательными являются только спецификаторы Select и From. Эти два спецификатора составляют основу каждого запроса к базе данных, поскольку они определяют таблицы, из которых выбираются данные, и столбцы, которые требуется выбрать.

Спецификатор Where добавляется для выборки определенных строк или указания условия соединения. Спецификатор Order by добавляется для изменения порядка получаемых данных. Спецификатор Into temp добавляется для сохранения этих результатов в виде таблицы с целью выполнения последующих запросов. Два дополнительных спецификатора оператора Select - Group by (спецификатор группирования) и Having (спецификатор условия выборки группы) - позволяют выполнять более сложные выборки данных.

У п р а ж н е н и я

1. Выбор всех строк и столбцов таблицы.

Пример .

Выдать полную информацию о поставщиках.

Select * from S

Результат: таблица S в полном объеме.

Подготовьте запрос и проверьте полученный результат.

2. Измение порядка следования столбцов.

Пример .

Выдать таблицу S в следующем порядке: фамилия, город, рейтинг, номер_поставщика.

Select фамилия, город, рейтинг, номер_поставщика from S

Результат: таблица S в требуемом порядке.

Подготовьте запрос и проверьте полученный результат.

3. Выбор заданных

столбцов. **Пример .**

Выдать номера всех поставляемых деталей.

Select номер_детали from SPJ Результат:

столбец номер_детали таблицы SPJ Подготовьте

запрос и проверьте полученный результат.

4. Выбор без

повторения. **Пример .**

Выдать номера всех поставляемых деталей, исключая дублирование.

Select distinct номер_детали from SPJ

Результат:	**номер_детали**
	P1
	P2
	P3
	P4
	P5
	P6

Подготовьте запрос и проверьте полученный результат.

5. Использование в запросах констант и

выражений. **Пример .**

Select номер_детали, "вес в граммах", вес*454 from P

Результат:	P1 вес в граммах=5448

	P6 вес в граммах=8226

Подготовьте запрос и проверьте полученный результат.

6.Ограничение в выборке.

Пример .

Выдать номера всех поставщиков, находящихся в Париже с рейтингом > 20.

Select номер_поставщика from S where город="Париж" and рейтинг>20

Результат:	номер_поставщика
	S3

Подготовьте запрос и проверьте полученный результат.

7. Выборка с

упорядочиванием. **Пример .**

Выдать номера поставщиков, находящихся в Париже в порядке убывания рейтинга.

Select номер_поставщика, рейтинг from S where город="Париж" order by рейтинг desc

Результат:	номер_поставщика	рейтинг
	S3	30
	S2	10

Подготовьте запрос и проверьте полученный результат.

8. Упорядочивание по нескольким

столбцам. **Пример .**

Выдать список поставщиков, упорядоченных по городу, в пределах города - по рейтингу.

Select * from S order by 4, 3

Результат:	Номер_поставщика	Фамилия	Рейтинг	Город
	S5	Адамс	30	Атенс
	S1	Смит	20	Лондон
	S4	Кларк	20	Лондон
	S2	Джонс	10	Париж
	S3	Блейк	30	Париж

Подготовьте запрос и проверьте полученный результат.

9. Фраза in (not in

). **Пример .**

Выдать детали, вес которых равен 12, 16 или 17.

Select номер_детали, название, вес from P where вес in (12, 16, 17)

Результат:	номер_детали	Название	вес
	P1	Гайка	12
	P2	Болт	17
	P3	Винт	17
	P5	Кулачок	12

Подготовьте запрос и проверьте полученный результат.

12. Выбор по шаблону.

Для запросов с поиском по шаблону, основанных на поиске подстрок в полях типа CHARACTER, используются ключевое слова LIKE.

Включение в выражение ключевого слова NOT порождает условие с обратным смыслом. Ключевое слово LIKE соответствует стандарту ANSI.

СИМВОЛ	ЗНАЧЕНИЕ
LIKE	
%	Заменяет последовательность символов
-	Заменяет любой одиночный символ
\	Отменяет специальное назначение следующего за ним симво-ла

Примеры.

а) Выбрать список деталей, начинающихся с буквы "Б"[10]

Select номер_детали, название, вес from P where название like "Б%"

Результат:	номер_детали	название	вес
	P5	Болт	12
	P6	Блюм	19

II. Использование функций

1. Агрегатные функции.

Примеры.

а) Выдать общее количество поставщиков.

Select count (*) from S

Результат: 5

Подготовьте запрос и проверьте полученный результат.

б) Выдать общее количество поставщиков, поставляющих в настоящее время детали.

Select count (distinct номер_поставщика) from

SPJ Результат: 4

Подготовьте запрос и проверьте полученный результат.

в) Выдать количество поставок для детали P2.

Select count (*) from SPJ where

номер_детали='P2' Результат: 5

Подготовьте запрос и проверьте полученный результат.

г) Выдать общее количество поставляемых деталей 'P2'.

Select sum (количество) from SPJ where номер_детали='P2'

Результат: 1000

[10] Примечание. Корректно работает только при задании кодировки по умолчанию. Задается в разделе MYSQLD default_character_set=win1251

Подготовьте запрос и проверьте полученный результат.

д) Выдать средний, минимальный и максимальный объем поставок для поставщика S1 с соответствующим заголовком.

Select avg(количество) average, min(количество) minimum,

max(количество) maximum from SPJ where номер_поставщика='S1'

Результат:	average	minimum	maximum
	216.6	100	400

Подготовьте запрос и проверьте полученный результат.

2. Ниже привиден перечень всех функций, используемых в операторе Select

Функции

select_expression может содержать следующие функции и операторы:

+ - * /	Арифметические действия.
%	Остаток от деления (как в C)
| &	Битовые функции (используется 48 бит).
- C	Мена знака числа.
( )	Скобки.
BETWEEN(A, B, C)	(A >= B) AND (A <= C).
BIT_COUNT()	Количество бит.
ELT(N, a, b, c, d)	Возвращает a, если N == 1, b, если N == 2 и т. д. a, b, c, d строки. **ПРИМЕР:** ELT(3, "First", "Second", "Third", "Fourth") вернет "Third".
FIELD(Z, a, b, c)	Возвращает a, если Z == a, b, если Z == b и т. д. a, b, c, d строки. **ПРИМЕР:** FIELD("Second", "First", "Second", "Third", "Fourth") вернет "Second".
IF(A, B, C)	Если A истина (!= 0 and != NULL), то вернет B, иначе вернет C.
IFNULL(A, B)	Если A не null, вернет A, иначе вернет B.
ISNULL(A)	Вернет 1, если A == NULL, иначе вернет 0. Эквивалент ('A == NULL').
NOT !	NOT, вернет TRUE (1) или FALSE (0).
OR, AND	Вернет TRUE (1) или FALSE (0).
SIGN()	Вернет -1, 0 или 1 (знак аргумента).
SUM()	Сумма столбца.
= <> <= < >= >	Вернет TRUE (1) или FALSE (0).
expr LIKE expr	Вернет TRUE (1) или FALSE (0).
expr NOT LIKE expr	Вернет TRUE (1) или FALSE (0).

expr REGEXP expr	Проверяет строку на соответствие регулярному выражению expr.

select_expression может также содержать один или большее количество следующих математических функций.

ABS()	Абсолютное значение (модуль числа).
CEILING()	()
EXP()	Экспонента.
FORMAT(nr, NUM)	Форматирует число в формат '#, ###, ###.##' с NUM десятичных цифр.
LOG()	Логарифм.
LOG10()	Логарифм по основанию 10.
MIN(), MAX()	Минимум или максимум соответственно. Должна иметь при вызове два или более аргументов, иначе рассматривается как групповая функция.
MOD()	Остаток от деления (аналог %).
POW()	Степень.
ROUND()	Округление до ближайшего целого числа.
RAND([integer_expr])	Случайное число типа float, 0 <= x <= 1.0, используется integer_expr как значение для запуска генератора.
SQRT()	Квадратный корень.

select_expression может также содержать одну или больше следующих строковых функций.

CONCAT()	Объединение строк.
INTERVAL(A, a, b, c, d)	Возвращает 1, если A == a, 2, если A == b... Если совпадений нет, вернет 0. A, a, b, c, d... строки.
INSERT(org, strt, len, new)	Заменяет подстроку org[strt...len(gth)] на new. Первая позиция строки=1.
LCASE(A)	Приводит A к нижнему регистру.
LEFT()	Возвращает строку символов, отсчитывая слева.
LENGTH()	Длина строки.
LOCATE(A, B)	Позиция подстроки B в строке A.
LOCATE(A, B, C)	Позиция подстроки B в строке A, начиная с позиции C.
LTRIM(str)	Удаляет все начальные пробелы из строки str.
REPLACE(A, B, C)	Заменяет все подстроки B в строке A на подстроку C.
RIGHT()	Get string counting from right.
RTRIM(str)	Удаляет хвостовые пробелы из строки str.
STRCMP()	Возвращает 0, если строки одинаковые.
SUBSTRING(A, B, C)	Возвращает подстроку из A, с позиции B до позиции C.
UCASE(A)	Переводит A в верхний регистр.

Еще несколько просто полезных функций, которые тоже можно применить в select_expression.

CURDATE()	Текущая дата.
DATABASE()	Имя текущей базы данных из которой выполняется выбор.
FROM_DAYS()	Меняет день на DATE.
NOW()	Текущее время в форматах YYYYMMDDHHMMSS или

	"YYYY-MM-DD HH:MM:SS". Формат зависит от того в каком контексте используется NOW(): числовом или строковом.
PASSWORD()	Шифрует строку.
PERIOD_ADD(P:N)	Добавить N месяцев к периоду P (в формате YYMM).
PERIOD_DIFF(A, B)	Возвращает месяцы между A и B. Обратите внимание, что PERIOD_DIFF работает только с датами в форме YYMM или YYYMM.
TO_DAYS()	Меняет DATE (YYMMDD) на номер дня.
UNIX_TIMESTAMP([date])	Возвращает метку времени unix, если вызвана без date (секунды, начиная с GMT 1970.01.01 00:00:00). При вызове со столбцом TIMESTAMP вернет TIMESTAMP. date может быть также строкой DATE, DATETIME или числом в формате YYMMDD (или YYYMMDD).
USER()	Возвращает логин текущего пользователя.
WEEKDAY()	Возвращает день недели (0 = понедельник, 1 = вторник, ...).

Групповые функции в операторе ***select*:**

Следующие функции могут быть использованы в предложении GROUP:

AVG()	Среднее для группы GROUP.
SUM()	Сумма элементов GROUP.
COUNT()	Число элементов в GROUP.
MIN()	Минимальный элемент в GROUP.
MAX()	Максимальный элемент в GROUP.

Задание:

1. Выполнить проверку запросов из:
   - 1го раздела (2, 6, 7, 9, 12), 2го раздела (а, б, д)
2. Подготовить 3 запроса с использованием различных функций работа с полем дата, со строковыми данными (в том числе групповых).
3. Подготовить и выполнить средствами СУБД MySQL 4 запроса по выборке информации из таблиц базы данных с использованием агрегатных функций..
4. Подготовить и выполнить средствами СУБД MySQL 2 запроса по модификации информации (вставка, удаление, замещение) из таблиц базы данных для решения нижеприведенных задач. При этом в тех заданиях, где речь идет о создании таблиц, предполагается формировании постоянной таблицы базы данных.

Варианты заданий на составление запросов по выборке информации из таблиц базы данных

Вариант 1.

1. Для каждой поставляемой для некоторого изделия детали выдать ее номер, номер изделия и соответствующее общее поставляемое количество деталей.
2. Выдать все триплеты "номер поставщика, номер детали и номер изделия", такие, что в каждом триплете указанные поставщик, деталь и изделие не являются попарно соразмещенными в одном городе.
3. Выдать номера изделий, для которых детали полностью поставляет поставщик S1. Т.е. поставляемых поставщиком S1 деталей достаточно для полного комплектования изделия. Состав деталей изделия можно оценить на основе базового набора данных таблицы поставка, имея в виду что в базовом наборе данных отражен полный состав всех изделий.
4. Выдать номера и фамилии поставщиков, поставляющих детали для какого-либо изделия с деталью P1 в количестве, большем, чем средний объем поставок детали P1 для этого изделия.

Вариант 2.

1. Выдать общее количество деталей P1, поставляемых поставщиком S1.
2. Выдать все пары названий городов, таких, что какой-либо поставщик из первого города поставляет детали для некоторого изделия, изготовляемого во втором городе.
3. Выдать номера изделий, использующих только детали, поставляемые поставщиком S1.
4. Выдать номера деталей, поставляемых каким-либо поставщиком из Лондона, для изделия, изготавливаемого также в Лондоне.

Вариант 3.

1. Выдать номера и фамилии поставщиков, поставляющих одну и ту же деталь для всех перечисленных изделий. Перечень изделий согласовать с преподавателем.
2. Выдать общее число изделий (не деталей), для которых поставляет детали поставщик S1.
3. Выдать номера изделий, детали для которых поставляет каждый поставщик, поставляющий какую-либо красную деталь. Т.е. необходимо получить такие номера изделий, детали для которой поставляются всеми поставщиками, среди поставляемых деталей которого есть детали красного цвета.
4. Выдать все триплеты "номер поставщика, номер детали и номер изделия", такие, что в каждом триплете указанные поставщик, деталь и изделие являются попарно соразмещенными в одном городе.

Вариант 4.

1. Выдать номера и фамилии поставщиков, поставляющих по крайней мере одну деталь, поставляемую по крайней мере одним поставщиком, который поставляет по крайней мере одну красную деталь. Т.е. необходимо выдать полные сведения о всех поставщиках которые поставляют такие детали, которые есть в поставках поставщиков, поставляющих красные детали.

2. Выдать список деталей, поставляющихся для **всех** изделий, изготавливаемых в Лондоне.
3. Выдать номера деталей, поставляемых каким-либо поставщиком из Лондона.
4. Выдать номера деталей, поставляемых для **какого-либо** изделия из Лондона.

Вариант 5.

1. Выдать номера изделий, для которых детали поставляются по крайней мере одним поставщиком не из того же самого города, что и изделие.
2. Выдать список всех поставок, в которых количество деталей находится в диапазоне от 300 до 750 включительно.
3. Выдать номера изделий, использующих, по крайней мере, одну деталь, поставляемую поставщиком S1. Т.е. показать такие изделия, для производства которых пригодились бы детали, поставляемые поставщиком S1.
4. Выдать номера и названия деталей, поставляемых для какого-либо изделия в Лондоне.

Варианты заданий на составление запросов по модификации информации из таблиц базы данных

Вариант 1.

1. Увеличить на 10 рейтинг всех поставщиков, рейтинг которых в настоящее время меньше, чем рейтинг поставщика S4.
2. Постройте таблицу, содержащую список номеров изделий, которые либо находятся в Лондоне, либо для них поставляются детали каким-нибудь поставщиком из Лондона.

Вариант 2.

1. Удалить все изделия, для которых нет поставок деталей.
2. Построить таблицу с номерами поставщиков и парами номеров деталей, таких, что некоторый поставщик поставляет обе указанные детали. При этом пары вида P1 и P2, а также P2 и P1 считать одинаковыми.

Вариант 3.

1. Увеличить размер поставки на 10 процентов для всех поставок тех поставщиков, которые поставляют какую-либо красную деталь.
2. Построить таблицу с комбинациями "цвет детали-город, где хранится деталь", исключая дубликаты пар (цвет-город).

Вариант 4.

1. Построить таблицу, содержащую список номеров деталей, которые поставляются либо каким-нибудь поставщиком из Лондона, либо для какого-либо изделия в Лондон.
2. Вставить в таблицу S нового поставщика с номером S10 с фамилией Уайт из города Нью-Йорк с неизвестным рейтингом.

Вариант 5.

1. Удалить все изделия из Рима и все соответствующие поставки.
2. Построить таблицу с упорядоченным списком всех городов, в которых размещаются по крайней мере один поставщик, деталь или изделие.

Контрольные вопросы

1. Что такое коррелированный запрос? Чем отличается коррелированный запрос от некоррелированного?
2. Какие существуют ограничения на формирование коррелированного запроса?
3. Каким образом сохранить результаты запроса в таблице?
4. Какими средствами SQL реализуются следующие операции реляцион-ной алгебры: ограничение, декартово произведение, проекция, пересе-чение, объединение, разность, соединение?
5. Что такое внешнее соединение?
6. В каких случаях вместо фразы IN можно использовать операцию сравне-ния?
7. Какие существуют средства группирования в SQL? Как они использу-ются?

Лабораторная работа 4
Работа с внешними базами данных. Ограничение доступа

Цель работы: Ознакомиться со средствами предоставления полномочий на использование баз данных и таблиц и основами работы с внешними базами данных.

Предоставление доступа к базам данных

СУБД MySQL использует специальную базу данных для предоставления прав доступа к своим базам данных. Эти права могут базироваться на именах серверов и/или пользователей и предоставляться для одной или нескольких баз данных

Пользовательские учетные записи могут быть снабжены паролями. При обращении к базе данных, пароль шифруется. Поэтому он не может быть перехвачен и использован посторонним (это мнение автора СУБД...). СУБД MySQL имеет три таблицы, а именно:

База данных: mysql Таблица: db

База данных: mysql Таблица: db					
Поле	**Тип**	**Null**	**Ключ**	**Умолчание**	**Extra**
Хост	char(60)		PRI		
Db	char(32)		PRI		
Пользователь	char(16)		PRI		
Select_priv	char(1)			N	
Insert_priv	char(1)			N	
Update_priv	char(1)			N	
Delete_priv	char(1)			N	
Create_priv	char(1)			N	
Drop_priv	char(1)			N	

База данных: mysql Таблица: host

Поле	**Тип**	**Null**	**Ключ**	**Умолчание**	**Extra**
Хост	char(60)		PRI		
Db	char(32)		PRI		
Select_priv	char(1)			N	
Insert_priv	char(1)			N	
Update_priv	char(1)			N	
Delete_priv	char(1)			N	
Create_priv	char(1)			N	
Drop_priv	char(1)			N	

База данных: mysql Таблица: user

Поле	Тип	Null	Key	Умолчание	Extra
Хост	char(60)		PRI		
Пользователь	char(16)		PRI		
Пароль	char(8)				
Select_priv	char(1)			N	
Insert_priv	char(1)			N	
Update_priv	char(1)			N	
Delete_priv	char(1)			N	
Create_priv	char(1)			N	
Drop_priv	char(1)			N	
Reload_priv	char(1)			N	
Shutdown_priv	char(1)			N	
Process_priv	char(1)			N	
File_priv	char(1)			N	

Текущей базой данных называется база данных, открытая с помощью операторов use Database или с помощью утилиты mysqladmin. Любая другая база данных называется внешней. Для ссылки на таблицу во внешней базе данных необходимо указать имя этой базы данных как часть имени таблицы, например, salesdb:contracts, где salesdb - имя внешней базы данных, contracts - имя таблицы. К имени базы данных можно добавить имя сервера, т.е. сете-вой машины, где запущен еще один сервер баз данных mysql, и таким об-разом в случае распределенной базы данных обращение к таблице contracts базы данных salesdb, размещенной на сервере central, будет выглядеть следу-ющим образом: salesdb@central:contracts.

В программе MYSQL-FRONT также существует механизм, обеспечивающий наделение пользователей определенными правами (рис. 4.1).

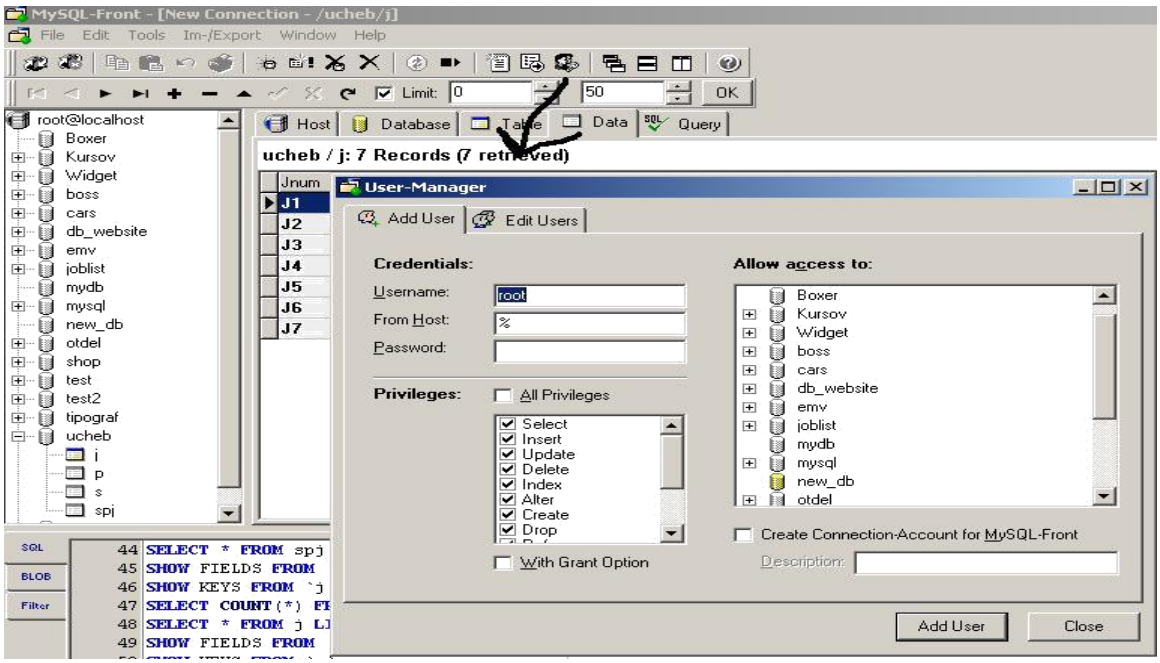

Рисунок 4.1 - Редактирование прав пользователя

При наличии сетевого соединения с сервером выполните приведенную последовательность выполнения лабораторной работы, при отсутствии соединения создать еще 1го пользователя в вашей БД, наделив его привилегиями лишь для просмотра таблиц, в этом случае все приведенные операции осуществлять от имени созданного пользователя.

Задание (общее):

1. Убедиться, что в таблице поставщиков S имеются строки с Вашими фамилиями (задание выполнялось в третьей лабораторной работе).
2. Откорректировать экранную форму, созданную в третьей лабораторной работе для работы с таблицей поставок SPJ, обеспечив возможность ввода и модификации данных. Занести произвольным образом несколько строк (5-10 строк) о поставках, связанных с Вашими фамилиями.
3. Выполнить два запроса к базе данных, согласно номера Вашего варианта. При выполнении запроса данные должны выбираться из таблиц Вашей базы данных.
4. Повторить задание п.3 с той разницей, что сведения о номенклатуре деталей и изделий (P и J) должна браться из собственной базы данных, а сведения о поставщиках и поставках (S и SPJ) должны браться из базы данных соседней бригады. Предварительно необходимо узнать имя этой базы данных. Убедитесь в невозможности выполнения задания.
5. Обеспечьте, чтобы владелец внешней используемой Вами базы данных предоставил Вам полномочия на просмотр используемых Вами таблиц в его базе данных, дав соответственно ему такие же полномочия для выполнения аналогичных действий.

6. Повторите задание п.4. Сравните результаты с результатами, полученными в п.3.
7. Сделайте попытку изменить информацию о поставщиках-владельцах базы данных (город, рейтинг и т.д.) в таблице S внешней базы данных. Убедитесь в невозможности выполнения задания.
8. Обеспечьте, чтобы владелец внешней используемой Вами базы данных предоставил Вам полномочия на модификацию данных из используемых Вами таблиц в его базе данных, дав соответственно ему такие же полномочия для выполнения аналогичных действий.
9. Повторите задание п.7. Проверьте успешность выполнения действий.
10. Дождавшись, когда владелец внешней базы данных закончит выполнение п.9, сделайте попытку удалить из таблицы S используемой Вами внешней базы данных поставщиков с именами, принадлежащими владельцам базы дан-ных, и связанные с ними поставки из таблицы SPJ. Убедитесь в невозможно-сти выполнения задания.
11. Обеспечьте, чтобы владелец используемой Вами внешней базы данных предоставил Вам полномочия на удаление из используемых Вами таблиц в его базе данных, дав соответственно ему такие же полномочия для выполнения аналогичных действий.
12. Повторите задание п.10. Проверьте успешность выполнения действий.
13. Отнимите предоставленные Вами права на пользование Вашей базой данных.

Индивидуальные варианты заданий

Вариант 1.

1. Выдать список всех поставок, в которых количество деталей находится в диапазоне от 300 до 750 включительно.
2. Выдать номера изделий, использующих по крайней мере одну деталь, поставляемую поставщиком S6.

Вариант 2.

1. Выдать цвета деталей, поставляемых поставщиком S6.
2. Выдать номера и фамилии поставщиков, поставляющих деталь P1 для какого-либо изделия в количестве, большем среднего объема поставок детали P1 для этого изделия.

Вариант 3.

1. Выдать названия изделий, для которых поставляются детали поставщиком S6.
2. Выдать номера и названия изделий, для которых поставщик S6 поставляет несколько деталей каждого из поставляемых им типов.

Вариант 4.

1. Для каждой поставляемой для некоторого изделия детали выдать ее номер, номер изделия и соответствующее общее количество деталей.
2. Выдать номера изделий, для которых детали полностью поставляет поставщик S6.

Вариант 5.

1. Выдать номера и фамилии поставщиков, поставляющих детали для какого-либо изделия с деталью P1 в количестве, большем, чем средний объем поставок детали P1 для этого изделия.
2. Выдать номера изделий, использующих только детали, поставляемые поставщиком S6.

Контрольные вопросы

1. Кто является владельцем базы данных?
2. Какими правами обладают другие пользователи по отношению к Вашей базе данных?
3. Какими правами обладает администратор базы данных по отношению к Вашей базе данных?
4. Каким образом предоставляются права на пользование базой данных и отдельными ее таблицами?
5. Каким образом изымаются права на пользование базой данных и отдельными ее таблицами?
6. Что такое внешняя база данных?
7. Как идентифицируется таблица внешней базы данных?
8. Как идентифицируется таблица внешней распределенной базы данных?

ПРИЛОЖЕНИЕ

Синтаксис оператора *CREATE TABLE*

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [(create_defini-tion,...)] [table_options] [select_statement]

create_definition:

col_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]
[PRIMARY KEY] [reference_definition]
или PRIMARY KEY (index_col_name,...)
или KEY [index_name] (index_col_name,...)
или INDEX [index_name] (index_col_name,...)
или UNIQUE [INDEX] [index_name] (index_col_name,...)
или FULLTEXT [INDEX] [index_name] (index_col_name,...)
или [CONSTRAINT symbol] FOREIGN KEY [index_name] (index_col_name,...)
[reference_definition]
или CHECK (expr)

type:
TINYINT[(length)] [UNSIGNED] [ZEROFILL]
или SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
или MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
или INT[(length)] [UNSIGNED] [ZEROFILL]
или INTEGER[(length)] [UNSIGNED] [ZEROFILL]
или BIGINT[(length)] [UNSIGNED] [ZEROFILL]
или REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
или DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
или FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
или DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
или NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
или CHAR(length) [BINARY]
или VARCHAR(length) [BINARY]
или DATE
или TIME
или TIMESTAMP
или DATETIME
или TINYBLOB
или BLOB
или MEDIUMBLOB
или LONGBLOB
или TINYTEXT
или TEXT
или MEDIUMTEXT
или LONGTEXT
или ENUM(value1,value2,value3,...)
или SET(value1,value2,value3,...)

index_col_name:

col_name [(length)]

reference_definition:
REFERENCES tbl_name [(index_col_name,...)]
[MATCH FULL | MATCH PARTIAL]
[ON DELETE reference_option]
[ON UPDATE reference_option]
reference_option:
RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
table_options:
TYPE = {BDB | HEAP | ISAM | InnoDB | MERGE | MRG_MYISAM |
MYISAM }

Оператор CREATE TABLE создает таблицу с заданным именем в текущей базе данных.

Для всех имен баз данных, таблиц, столбцов, индексов и псевдонимов в MySQL приняты одни и те же правила.

Следует отметить, что эти правила были изменены, начиная с версии MySQL 3.23.6, когда было разрешено брать в одиночные скобки (') идентификаторы (имена баз данных, таблиц и столбцов). Двойные скобки (") тоже допустимы - при работе в режиме ANSI SQL.

Идентификатор	Макс. длина строки	Допускаемые символы
База данных	64	Любой символ, допустимый в имени каталога, за исключением (/) или (.)
Таблица	64	Любой символ, допустимый в имени файла, за исключением (/) или (.)
Столбец	64	Все символы
Псевдоним	255	Все символы

Если нет активной текущей базы данных или указанная таблица уже существует, то возникает ошибка выполнения команды.

В версии MySQL 3.22 и более поздних имя таблицы может быть указано как db_name.tbl_name. Эта форма записи работает независимо от того, является ли указанная база данных текущей.

В версии MySQL 3.23 при создании таблицы можно использовать ключевое слово TEMPORARY. Временная таблица автоматически удаляется по завершении соединения, а
ее имя действительно только в течение данного соединения. Это означает, что в двух разных соединениях могут использоваться временные таблицы с одинаковыми именами без конфликта друг с другом или с существующей таблицей с тем же именем (существующая таблица скрыта, пока не удалена временная таблица). В версии MySQL 4.0.2 для создания временных таблиц необходимо иметь привилегии CREATE TEMPORARY TABLES.

В версии MySQL 3.23 и более поздних можно использовать ключевые слова IF NOT EXISTS для того, чтобы не возникала ошибка, если указанная таблица уже существует. Следует учитывать, что при этом не проверяется идентичность структур этих таблиц.

Каждая таблица tbl_name представлена определенными файлами в директории базы данных. В случае таблиц типа MyISAM - это следующие файлы:

Файл	Назначение
tbl_name.frm	Файл определения таблицы
tbl_name.MYD	Файл данных
tbl_name.MYI	Файл индексов

Чтобы получить более полную информацию о свойствах различных типов столбцов, см. документацию к СУБД.

Если не указывается ни NULL, ни NOT NULL, то столбец интерпретируется так, как будто указано NULL.

Целочисленный столбец может иметь дополнительный атрибут AUTO_INCREMENT. При записи величины NULL (рекомендуется) или 0 в столбец AUTO_INCREMENT данный столбец устанавливается в значение value+1, где value представляет собой наибольшее для этого столбца значение в таблице на момент записи. Последовательность AUTO_INCREMENT начинается с 1. Если удалить строку, содержащую максимальную величину для столбца AUTO_INCREMENT, то в таблицах типа ISAM или BDB эта величина будет восстановлена, а в таблицах типа MyISAM или InnoDB - нет. Если удалить все строки в таблице командой DELETE FROM table_name (без выражения WHERE) в режиме AUTOCOMMIT, то для таблиц всех типов последовательность начнется заново.

Примечание: в таблице может быть только один столбец AUTO_INCREMENT, и он должен быть индексирован. Кроме того, версия MySQL 3.23 будет правильно работать только с положительными величинами столбца AUTO_INCREMENT. В случае внесения отрицательного числа оно интерпретируется как очень большое положительное число. Это делается, чтобы избежать проблем с точностью, когда числа "заворачиваются" от положительного к отрицательному и, кроме того, для гарантии, что по ошибке не будет получен столбец AUTO_INCREMENT со значением 0.

Величины NULL для столбца типа TIMESTAMP обрабатываются иначе, чем для столбцов других типов. В столбце TIMESTAMP нельзя хранить литерал NULL; при установке данного столбца в NULL он будет установлен в текущее значение даты и времени. Поскольку столбцы TIMESTAMP ведут себя подобным образом, то атрибуты NULL и NOT NULL неприменимы в обычном режиме и игнорируются при их задании. С другой стороны, чтобы облегчить клиентам MySQL использование столбцов TIMESTAMP, сервер сообщает, что таким столбцам могут быть назначены величины NULL (что соответствует действительности), хотя реально TIMESTAMP никогда не будет содержать величины NULL. Это можно увидеть, применив DESCRIBE tbl_name для получения описания данной таблицы. Следует учитывать, что установка столбца TIMESTAMP в 0 не равно-значна установке его в NULL, поскольку 0 для TIMESTAMP является допустимой величиной.

Величина DEFAULT должна быть константой, она не может быть функцией или вы-ражением. Если для данного столбца не задается никакой величины DEFAULT, то MySQL автоматически назначает ее. Если столбец может принимать NULL как допустимую ве-личину, то по умолчанию присваивается значение NULL. Если столбец объявлен как NOT NULL, то значение по умолчанию зависит от типа столбца: для числовых типов, за исключением объявленных с атрибутом AUTO_INCREMENT, значение по умолчанию равно 0. Для столбца AUTO_INCREMENT значением по умолчанию является следующее значение в последовательности для этого столбца.

Для типов даты и времени, отличных от TIMESTAMP, значение по умолчанию рав-но соответствующей нулевой величине для данного типа. Для первого столбца TIMESTAMP в таблице значение по умолчанию представляет собой текущее значение даты и времени.

Для строковых типов, кроме ENUM, значением по умолчанию является пустая строка. Для ENUM значение по умолчанию равно первой перечисляемой величине (если явно не задано другое значение по умолчанию с помощью директивы DEFAULT).

Значения по умолчанию должны быть константами. Это означает, например, что нельзя установить для столбца "даты" в качестве значения по умолчанию величину функции, такой как NOW() или CURRENT_DATE.

KEY является синонимом для INDEX.

В MySQL ключ UNIQUE может иметь только различающиеся значения. При попыт-
ке добавить новую строку с ключом, совпадающим с существующей строкой, возникает ошибка выполнения команды.

PRIMARY KEY представляет собой уникальный ключ KEY с дополнительным ограничением, т.е. столбцы с данным ключом должны быть определены как NOT NULL. В MySQL этот ключ называется PRIMARY (первичный). Таблица может иметь только один первичный ключ PRIMARY KEY. Если PRIMARY KEY отсутствует в таблицах, а неко-торое приложение запрашивает его, то MySQL может превратить в PRIMARY KEY пер-вый ключ UNIQUE, не имеющий ни одного столбца NULL.

PRIMARY KEY может быть многостолбцовым индексом. Однако нельзя создать многостолбцовый индекс, используя в определении столбца атрибут ключа PRIMARY KEY. Именно таким образом только один столбец будет отмечен как первичный. Необходи-мо использовать синтаксис PRIMARY KEY(index_col_name, ...).

Если ключ PRIMARY или UNIQUE состоит только из одного столбца и он принад-лежит к числовому типу, то на него можно сослаться также, как на _rowid (новшество
версии 3.23.11).

Если индексу не назначено имя, то ему будет присвоено первое имя в index_col_name, возможно, с суффиксами (_2, _3, ...), делающими это имя уни-
кальным. Имена индексов для таблицы можно увидеть, используя SHOW INDEX FROM tbl_name. SHOW Syntax.

С помощью выражения col_name(length) можно указать индекс, для которо-го используется только часть столбца CHAR или VARCHAR. Это поможет сделать файл индексов намного меньше.

Индексацию столбцов BLOB и TEXT поддерживают только таблицы с типом MyISAM. Назначая индекс столбцу с типом BLOB или TEXT, всегда НЕОБХОДИМО ука-зывать длину этого индекса:
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));

В версии MySQL 3.23.23 и более поздних можно создавать также специальные индексы FULLTEXT. Они применяются для полнотекстового поиска. Эти индексы поддерживаются только таблицами типа MyISAM, и они могут быть созданы только из столбцов VARCHAR и TEXT. Индексирование всегда выполняется для всего столбца целиком, частичная индексация не поддерживается. Более подробно эта операция описана в разделе MySQL section.

Выражения FOREIGN KEY, CHECK и REFERENCES фактически ничего не де-
лают. Они введены только из соображений совместимости, чтобы облегчить перенос кода с других SQL-серверов и запускать приложения, создающие таблицы со ссылками.

Для каждого столбца NULL требуется один дополнительный бит, при этом величина столбца округляется в большую сторону до ближайшего байта.

Максимальную длину записи в байтах можно вычислить следующим образом:
длина записи = 1+
+ (сумма длин столбцов)+

+ (количество столбцов с допустимым NULL + 7)/8+
+ (количество столбцов с динамической длинной).

Опции table_options и SELECT реализованы только в версиях MySQL 3.23 и выше. Ниже представлены различные типы таблиц:

Тип таблицы	**Описание**
BDB	Таблицы с поддержкой транзакций и блокировкой страниц
HEAP	Данные для этой таблицы хранятся только в памяти
ISAM	Оригинальный обработчик таблиц
InnoDB	Таблицы с поддержкой транзакций и блокировкой строк. See section
MERGE	Набор таблиц MyISAM, используемый как одна таблица. See section
MRG_MyISAM	Псевдоним для таблиц MERGE
MyISAM	Новый обработчик, обеспечивающий переносимость таблиц в бинарном виде, который заменяет ISAM. See section

Если задается тип таблицы, который не поддерживается данной версией, то MySQL выберет из возможных типов ближайший к указанному. Например, если задается TYPE=BDB и данный дистрибутив MySQL не поддерживает таблиц BDB, то вместо этого будет создана таблица MyISAM. Другие табличные опции используются для оптимизации характеристик таблицы. Эти опции в большинстве случаев не требуют специальной установки. Данные опции работают с таблицами всех типов, если не указано иное:

Опция	**Описание**
AUTO_INCREMENT	Следующая величина AUTO_INCREMENT, которую следует установить для данной таблицы (MyISAM)
AVG_ROW_LENGTH	Приближенное значение средней длины строки для данной таблицы. Имеет смысл устанавливать только для обширных таблиц с записями переменной длины
CHECKSUM	Следует установить в 1, чтобы в MySQL поддерживалась проверка контрольной суммы для всех строк (это делает таблицы немного более медленными при обновлении, но позволяет легче находить поврежденные таблицы) (MyISAM)
COMMENT	Комментарий для данной таблицы длиной 60 символов
MAX_ROWS	Максимальное число строк, которые планируется хранить в данной таблице
MIN_ROWS	Минимальное число строк, которые планируется хранить в данной таблице
PACK_KEYS	Следует установить в 1 для получения меньшего индекса. Обычно это замедляет обновление и ускоряет чтение (MyISAM, ISAM). Установка в 0 отключит уплотнение ключей. При установке в DEFAULT (MySQL 4.0) обработчик таблиц будет уплотнять только длинные столбцы CHAR/VARCHAR
PASSWORD	Шифрует файл `.frm' с помощью пароля. Эта опция не функционирует в стандартной версии MySQL
DELAY_KEY_WRITE	Установка в 1 задерживает операции обновления таблицы ключей, пока не закроется указанная таблица (MyISAM)
ROW_FORMAT	Определяет, каким образом должны храниться строки. В настоящее время эта опция работает только с таблицами MyISAM, которые поддерживают форматы строк DYNAMIC и FIXED

При использовании таблиц MyISAM MySQL вычисляет выражение max_rows * avg_row_length, чтобы определить, насколько велика будет результирующая табли-

ца. Если не задана ни одна из вышеупомянутых опций, то максимальный размер таблицы будет составлять 4Гб (или 2Гб, если данная операционная система поддерживает только таблицы величиной до 2Гб). Это делается для того, чтобы (если нет реальной необходимости в больших файлах), ограничить размеры указателей, что позволит сделать индексы меньше и быстрее. Если опция PACK_KEYS не используется, то по умолчанию уплотняются толь-ко строки, но не числа. При использовании PACK_KEYS=1 числа тоже будут уплотнять-
ся. При уплотнении двоичных числовых ключей MySQL будет использовать сжатие префиксов. Это означает, что выгода от этого будет значительной только в случае большого количества одинаковых чисел. При сжатии префиксов для каждого ключа требуется один дополнительный байт, в котором указано, сколько байтов предыдущего ключа являются такими же, как и для следующего (следует учитывать, что указатель на строку хранится в порядке "старший- байт-в-начале", сразу после ключа, - чтобы улучшить компрессию). Это означает, что при наличии нескольких одинаковых ключей в двух строках записи все последующие "аналогичные" ключи будут занимать только по 2 байта (включая указатель строки). Сравним: в обычном случае для хранения последующих ключей требуется
размер_хранения_ключа + размер_указателя (обычно 4) байтов.
С другой стороны, если все ключи абсолютно разные, каждый ключ будет занимать на 1 байт больше, если данный ключ не может иметь величину NULL (в этом случае уплотненный ключ будет храниться в том же байте, который используется для указания, что ключ равен NULL).

Если после команды CREATE указывается команда SELECT, то MySQL создаст новые поля для всех элементов в данной команде SELECT. Например:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
           PRIMARY KEY (a), KEY(b))
           TYPE=MyISAM SELECT b,c FROM test2;
```

Эта команда создаст таблицу MyISAM с тремя столбцами: a, b и c. Отметим, что столбцы из команды SELECT присоединяются к таблице справа, а не перекрывают ее. Рассмотрим следующий пример:

```
mysql> SELECT * FROM foo;
+---+
| n |
+---+
| 1 |
+---+
mysql> CREATE TABLE bar (m INT) SELECT n FROM foo; Query
OK, 1 row affected (0.02 sec)
Records: 1 Duplicates: 0 Warnings: 0 mysql>
SELECT * FROM bar; +------+---+
| m          | n |
+------+---+
| NULL | 1 |
+------+---+
1 row in set (0.00 sec)
```

Каждая строка в таблице foo вносится в таблицу bar со своим значением из foo, при этом в новые столбцы в таблице bar записываются величины, заданные по умолча-нию. Команда CREATE TABLE ... SELECT не создает автоматически каких-либо индексов. Это сделано преднамеренно, чтобы команда была настолько гибкой, насколько возможно. Чтобы иметь индексы в созданной таблице, необходимо указать их перед данной командой SELECT:

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

Если возникает ошибка при копировании данных в таблицу, то они будут автоматически удалены. Чтобы обеспечить возможность использовать для восстановления таблиц журнал обновлений/двоичный журнал, в MySQL во время выполнения команды CREATE TABLE ... SELECT не разрешены параллельные вставки.

Синтаксис оператора *ALTER TABLE*

ALTER [IGNORE] TABLE tbl_name alter_spec [, alter_spec ...] alter_specification:

```
        ADD [COLUMN] create_definition [FIRST | AFTER column_name]
или     ADD [COLUMN] (create_definition, create_definition,...)
или     ADD INDEX [index_name] (index_col_name,...)
или     ADD PRIMARY KEY (index_col_name,...)
или     ADD UNIQUE [index_name] (index_col_name,...)
или     ADD FULLTEXT [index_name] (index_col_name,...)
или     ADD [CONSTRAINT symbol] FOREIGN KEY index_name
(index_col_name,...)
              [reference_definition]
или     ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DE-
FAULT}
или     CHANGE [COLUMN] old_col_name create_definition
                [FIRST | AFTER column_name]
или     MODIFY [COLUMN] create_definition [FIRST | AFTER
column_name]
или     DROP [COLUMN] col_name
или     DROP PRIMARY KEY
или     DROP INDEX index_name
или     DISABLE KEYS
или     ENABLE KEYS
или     RENAME [TO] new_tbl_name
или     ORDER BY col
или     table_options
```

Оператор ALTER TABLE обеспечивает возможность изменять структуру существующей таблицы. Например, можно добавлять или удалять столбцы, создавать или уничтожать индексы или переименовывать столбцы либо саму таблицу. Можно также изменять комментарий для таблицы и ее тип.

Оператор ALTER TABLE во время работы создает временную копию исходной таблицы. Требуемое изменение выполняется на копии, затем исходная таблица удаляется, а новая переименовывается. Так делается для того, чтобы в новую таблицу автоматически попадали все обновления, кроме неудавшихся. Во время выполнения ALTER TABLE исходная таблица доступна для чтения другими клиентами. Операции обновления и записи в этой таблице приостанавливаются, пока не будет готова новая таблица.

Следует отметить, что при использовании любой другой опции для ALTER TABLE, кроме RENAME, MySQL всегда будет создавать временную таблицу, даже если данные, строго говоря, и не нуждаются в копировании (например, при изменении имени столбца). Для таблиц MyISAM можно увеличить скорость воссоздания индексной части (что является наиболее медленной частью в процессе восстановления таблицы) путем установки переменной myisam_sort_buffer_size достаточно большого значения.

Для использования оператора ALTER TABLE необходимы привилегии ALTER, INSERT и CREATE для данной таблицы.

Опция IGNORE является расширением MySQL по отношению к ANSI SQL92. Она управляет работой ALTER TABLE при наличии дубликатов уникальных ключей в новой таблице. Если опция IGNORE не задана, то для данной копии процесс прерывается и происходит откат назад. Если IGNORE указывается, тогда для строк с дубликатами уникальных ключей только первая строка используется, а остальные удаляются.

Можно запустить несколько выражений ADD, ALTER, DROP и CHANGE в одной команде ALTER TABLE. Это является расширением MySQL по отношению к ANSI SQL92, где допускается только одно выражение из упомянутых в одной команде ALTER TABLE.

Опции CHANGE col_name, DROP col_name и DROP INDEX также являются расширениями MySQL по отношению к ANSI SQL92.

Опция MODIFY представляет собой расширение Oracle для команды ALTER TABLE.

Необязательное слово COLUMN представляет собой "белый шум" и может быть опущено.

При использовании ALTER TABLE имя_таблицы RENAME TO новое_имя без каких-либо других опций MySQL просто переименовывает файлы, соответствующие заданной таблице. В этом случае нет необходимости создавать временную таблицу. В выражении create_definition для ADD и CHANGE используется тот же синтаксис, что и для CREATE TABLE. Следует учитывать, что этот синтаксис включает имя столбца, а не просто его тип.

Столбец можно переименовывать, используя выражение CHANGE имя_столб-ца create_definition. Чтобы сделать это, необходимо указать старое и новое имена столбца и его тип в настоящее время. Например, чтобы переименовать столбец INTEGER из a в b, можно сделать следующее:

```
mysql> ALTER TABLE t1 CHANGE a b INTEGER;
```

При изменении типа столбца, но не его имени синтаксис выражения CHANGE все равно требует указания обоих имен столбца, даже если они одинаковы. Например:

```
mysql> ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

Однако, начиная с версии MySQL 3.22.16a, можно также использовать выражение MODIFY для изменения типа столбца без переименовывания его:

```
mysql> ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

При использовании CHANGE или MODIFY для того, чтобы уменьшить длину столбца, по части которого построен индекс (например, индекс по первым 10 символам столбца VARCHAR), нельзя сделать столбец короче, чем число проиндексированных символов.

При изменении типа столбца с использованием CHANGE или MODIFY MySQL пы-тается преобразовать данные в новый тип как можно корректнее.

В версии MySQL 3.22 и более поздних можно использовать FIRST или ADD ... AFTER имя_столбца для добавления столбца на заданную позицию внутри табличной строки. По умолчанию столбец добавляется в конце. Начиная с версии MySQL 4.0.1, можно также использовать ключевые слова FIRST и AFTER в опциях CHANGE или MODIFY.

Опция ALTER COLUMN задает для столбца новое значение по умолчанию или удаляет старое. Если старое значение по умолчанию удаляется и данный столбец может принимать значение NULL, то новое значение по умолчанию будет NULL. Если столбец не может быть NULL, то MySQL назначает значение по умолчанию. Опция DROP INDEX удаляет индекс. Это является расширением MySQL по отношению к ANSI SQL92. Если

столбцы удаляются из таблицы, то эти столбцы удаляются также и из любого индекса, в который они входят как часть. Если все столбцы, составляющие индекс, удаляются, то данный индекс также удаляется.

Если таблица содержит только один столбец, то этот столбец не может быть удален. Вместо этого можно удалить данную таблицу, используя команду DROP TABLE.

Опция DROP PRIMARY KEY удаляет первичный индекс. Если такого индекса в данной таблице не существует, то удаляется первый индекс UNIQUE в этой таблице. (MySQL отмечает первый уникальный ключ UNIQUE как первичный ключ PRIMARY KEY, если никакой другой первичный ключ PRIMARY KEY не был явно указан). При до-бавлении UNIQUE INDEX или PRIMARY KEY в таблицу они хранятся перед остальными неуникальными ключами, чтобы можно было определить дублирующиеся ключи как можно раньше.

Опция ORDER BY позволяет создавать новую таблицу со строками, размещенными в заданном порядке. Следует учитывать, что созданная таблица не будет сохранять этот порядок строк после операций вставки и удаления. В некоторых случаях такая возможность может облегчить операцию сортировки в MySQL, если таблица имеет такое расположение столбцов, которое Вы хотели бы иметь в дальнейшем. Эта опция в основном полезна, если заранее известен определенный порядок, в котором преимущественно будут запрашиваться строки. Использование данной опции после значительных преобразований таблицы дает возможность получить более высокую производительность.

При использовании команды ALTER TABLE для таблиц MyISAM все неуникаль-ные индексы создаются в отдельном пакете (подобно REPAIR). Благодаря этому команда ALTER TABLE при наличии нескольких индексов будет работать быстрее.

Начиная с MySQL 4.0, вышеуказанная возможность может быть активизирована явным образом. Команда ALTER TABLE ... DISABLE KEYS блокирует в MySQL обновление неуникальных индексов для таблиц MyISAM. После этого можно применить команду ALTER TABLE ... ENABLE KEYS для воссоздания недостающих индексов. Так как MySQL делает это с помощью специального алгоритма, который намного быстрее в сравнении со вставкой ключей один за другим, блокировка ключей может дать существенное ускорение на больших массивах вставок.

Применяя функцию C API mysql_info(), можно определить, сколько записей было скопировано, а также (при использовании IGNORE) - сколько записей было удалено из-за дублирования значений уникальных ключей.

Выражения FOREIGN KEY, CHECK и REFERENCES фактически ничего не делают. Они введены только из соображений совместимости, чтобы облегчить перенос кода с других серверов SQL и запуск приложений, создающих таблицы со ссылками.

Ниже приводятся примеры, показывающие некоторые случаи употребления команды ALTER TABLE. Пример начинается с таблицы t1, которая создается следующим образом:

```
mysql> CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

Для того чтобы переименовать таблицу из t1 в t2:

```
mysql> ALTER TABLE t1 RENAME t2;
```

Для того чтобы изменить тип столбца с INTEGER на TINYINT NOT NULL (оставляя имя прежним) и изменить тип столбца b с CHAR(10) на CHAR(20) с переименованием его с b на c:

```
mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c
CHAR(20);
```

Для того чтобы добавить новый столбец TIMESTAMP с именем d:

```
mysql> ALTER TABLE t2 ADD d TIMESTAMP;
```

Для того чтобы добавить индекс к столбцу d и сделать столбец a первичным ключом:

```
mysql> ALTER TABLE t2 ADD INDEX (d), ADD PRIMARY KEY (a);
```

Для того чтобы удалить столбец c:

```
mysql> ALTER TABLE t2 DROP COLUMN c;
```

Для того чтобы добавить новый числовой столбец AUTO_INCREMENT с именем c:

```
mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT, ADD
    INDEX (c);
```

Заметьте, что столбец c индексируется, так как столбцы AUTO_INCREMENT должны быть индексированы; кроме того, столбец c объявляется как NOT NULL, поскольку индексированные столбцы не могут быть NULL.

При добавлении столбца AUTO_INCREMENT значения этого столбца автоматиче-ски заполняются последовательными номерами (при добавлении записей). Первый номер последовательности можно установить путем выполнения команды SET INSERT_ID=# перед ALTER TABLE или использования табличной опции AUTO_INCREMENT = #.

Синтаксис оператора *DROP TABLE, DATABASE*

```
DROP TABLE [IF EXISTS] tbl_name [, tbl_name,..] [RESTRICT CASCADE]
```

Оператор DROP TABLE удаляет одну или несколько таблиц. Все табличные данные и определения удаляются, так что будьте внимательны при работе с этой командой! Действия с БД аналогичны.

Оператор DROP DATABASE удаляет все таблицы в указанной базе данных и саму базу. Если Вы выполняете DROP DATABASE на базе данных, символически связанных с
другой, то удаляется как ссылка, так и оригинальная база данных. Будьте очень внимательны при работе с этой командой.

Синтаксис оператора *UPDATE*

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
SET col_name1=expr1 [, col_name2=expr2, ...][WHERE where_definition][LIMIT #]
```

Оператор UPDATE обновляет столбцы в соответствии с их новыми значениями в строках существующей таблицы. В выражении SET указывается, какие именно столбцы следует модифицировать и какие величины должны быть в них установлены. В выражении WHERE, если оно присутствует, задается, какие строки подлежат обновлению. В остальных
случаях обновляются все строки. Если задано выражение ORDER BY, то строки будут обновляться в указанном в нем порядке.

Если указывается ключевое слово LOW_PRIORITY, то выполнение данной команды UPDATE задерживается до тех пор, пока другие клиенты не завершат чтение этой таблицы.

Если указывается ключевое слово IGNORE, то команда обновления не будет прервана, даже если при обновлении возникнет ошибка дублирования ключей. Строки, из-за которых возникают конфликтные ситуации, обновлены не будут.

Если доступ к столбцу из указанного выражения осуществляется по аргументу tbl_name, то команда UPDATE использует для этого столбца его текущее значение.

Например, следующая команда устанавливает столбец age в значение, на единицу большее его текущей величины:

```
mysql> UPDATE persondata SET age=age+1;
```

Значения команда UPDATE присваивает слева направо. Например, следующая команда дублирует столбец age, затем инкрементирует его:

```
mysql> UPDATE persondata SET age=age*2, age=age+1;
```

Если столбец устанавливается в его текущее значение, то MySQL замечает это и не обновляет его.

Команда UPDATE возвращает количество фактически измененных строк. В версии MySQL 3.22 и более поздних функция C API mysql_info() возвращает количество строк, которые были найдены и обновлены, и количество предупреждений, имевших место при выполнении UPDATE.

Синтаксис оператора *DELETE*

```
DELETE [LOW_PRIORITY | QUICK] FROM table_name
        [WHERE where_definition]
        [ORDER BY ...]
        [LIMIT rows]
```

или

```
DELETE [LOW_PRIORITY | QUICK] table_name[.*] [,table_name[.*] ...]
        FROM table-references
        [WHERE where_definition]
```

или

```
DELETE [LOW_PRIORITY | QUICK]
        FROM table_name[.*], [table_name[.*] ...] USING table-
        references
        [WHERE where_definition]
```

Оператор DELETE удаляет из таблицы table_name строки, удовлетворяющие заданным в where_definition условиям, и возвращает число удаленных записей.

Если оператор DELETE запускается без определения WHERE, то удаляются все строки. При работе в режиме AUTOCOMMIT это будет аналогично использованию опера-тора TRUNCATE. В MySQL 3.23 оператор DELETE без определения WHERE возвратит
ноль как число удаленных записей.

Если действительно необходимо знать число удаленных записей при удалении всех строк и если допустимы потери в скорости, то можно использовать команду DELETE в следующей форме:

```
mysql> DELETE FROM table_name WHERE 1>0;
```

Следует учитывать, что эта форма работает намного медленнее, чем DELETE FROM table_name без выражения WHERE, поскольку строки удаляются поочередно
по одной.

Если указано ключевое слово LOW_PRIORITY, выполнение данной команды DELETE будет задержано до тех пор, пока другие клиенты не завершат чтение этой табли-цы.

Если задан параметр QUICK, то обработчик таблицы при выполнении удаления не будет объединять индексы - в некоторых случаях это может ускорить данную операцию.

Символы .* после имен таблиц требуются только для совместимости с Access:

```
DELETE t1,t2 FROM t1,t2,t3 WHERE t1.id=t2.id AND
t2.id=t3.id
```

или

```
DELETE FROM t1,t2 USING t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id
```

В предыдущем случае просто удалены совпадающие строки из таблиц t1 и t2.

Выражение ORDER BY и использование нескольких таблиц в команде DELETE поддерживается в MySQL 4.0.

Если применяется выражение ORDER BY, то строки будут удалены в указанном порядке. В действительности это выражение полезно только в сочетании с LIMIT. Напри-мер:

```
DELETE FROM somelog
        WHERE user = 'jcole'
```

```
ORDER BY timestamp
LIMIT 1
```

Данный оператор удалит самую старую запись (по timestamp), в которой строка соответствует указанной в выражении WHERE.

Специфическая для MySQL опция LIMIT для команды DELETE указывает серверу максимальное количество строк, которые следует удалить до возврата управления клиенту. Эта опция может использоваться для гарантии того, что данная команда DELETE не потребует слишком много времени для выполнения. Можно просто повторять команду DELETE до тех пор, пока количество удаленных строк меньше, чем величина LIMIT.

Синтаксис оператора

SELECT Оператор SELECT имеет следующую структуру:

```
SELECT [STRAIGHT_JOIN]
                [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUF-
          FER_RESULT]
                                [SQL_CACHE      |      SQL_NO_CACHE]
          [SQL_CALC_FOUND_ROWS] [HIGH_PRIORITY]
         [DISTINCT | DISTINCTROW | ALL]
      select_expression,...
           [INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
      [FROM table_references
        [WHERE where_definition]
         [GROUP BY {unsigned_integer | col_name | for-mula} [ASC |
          DESC], ...]
        [HAVING where_definition]
         [ORDER BY {unsigned_integer | col_name | for-mula} [ASC |
          DESC], ...]
        [LIMIT [offset,] rows]
        [PROCEDURE procedure_name]
        [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT применяется для извлечения строк, выбранных из одной или нескольких таблиц.

select_expression может содержать следующие функции и операторы:

+ - * /	Арифметические действия
%	Остаток от деления (как в C)
| &	Битовые функции (используется 48 бит)
- C	Мена знака числа
( )	Скобки
BETWEEN(A, B, C)	(A >= B) AND (A <= C)
BIT_COUNT()	Количество бит
ELT(N, a, b, c, d)	Возвращает a, если N == 1, b, если N == 2 и т. д. a, b, c, d - строки. ПРИМЕР: ELT(3, "First", "Second", "Third", "Fourth") вернет "Third"
FIELD(Z, a, b, c)	Возвращает a, если Z == a; b, если Z == b и т. д., где a, b, c, d строки ПРИМЕР: FIELD("Second", "First", "Second", "Third", "Fourth") вернет "Second"
IF(A, B, C)	Если A истина (!= 0 and != NULL), то вернет B, иначе вернет C

IFNULL(A, B)	Если A не null, вернет A, иначе вернет B
ISNULL(A)	Вернет 1, если A == NULL, иначе вернет 0. Эквивалент ('A == NULL')
NOT !	NOT, вернет TRUE (1) или FALSE (0)
OR, AND	Вернет TRUE (1) или FALSE (0)
SIGN()	Вернет -1, 0 или 1 (знак аргумента)
SUM()	Сумма столбца
= <> <= < >= >	Вернет TRUE (1) или FALSE (0)
expr LIKE expr	Вернет TRUE (1) или FALSE (0)
expr NOT LIKE expr	Вернет TRUE (1) или FALSE (0)
expr REGEXP expr	Проверяет строку на соответствие регулярному выражению expr
expr NOT REGEXP expr	Проверяет строку на соответствие регулярному выражению expr

select_expression может также содержать один или большее количество следующих математических функций:

ABS()	Абсолютное значение (модуль числа)
CEILING()	()
EXP()	Экспонента
FORMAT(nr, NUM)	Форматирует число в формат '#, ###, ###.##' с NUM десятичных цифр
LOG()	Логарифм
LOG10()	Логарифм по основанию 10
MIN(), MAX()	Минимум или максимум соответственно. Должна иметь при вызове два или более аргумента, иначе рассматривается как групповая функция
MOD()	Остаток от деления (аналог %)
POW()	Степень
ROUND()	Округление до ближайшего целого числа
RAND([integer_expr])	Случайное число типа float, 0 <= x <= 1.0, используется integer_expr как значение для запуска генератора
SQRT()	Квадратный корень

select_expression может также содержать одну или больше следующих строковых функций.

CONCAT()	Объединение строк
INTERVAL(A, a, b, c, d)	Возвращает 1, если A == a; 2, если A == b... Если совпадений нет, вернет 0. A, a, b, c, d - строки.
INSERT(org, strt, len, new)	Заменяет подстроку org[strt...len(gth)] на new. Первая позиция строки=1
LCASE(A)	Приводит A к нижнему регистру
LEFT()	Возвращает строку символов, отсчитывая слева
LENGTH()	Длина строки
LOCATE(A, B)	Позиция подстроки B в строке A
LOCATE(A, B, C)	Позиция подстроки B в строке A, начиная с позиции C
LTRIM(str)	Удаляет все начальные пробелы из строки str
REPLACE(A, B, C)	Заменяет все подстроки B в строке A на подстроку C
RIGHT()	Получение подстроки справа
RTRIM(str)	Удаляет хвостовые пробелы из строки str
STRCMP()	Возвращает 0, если строки одинаковые
SUBSTRING(A, B, C)	Возвращает подстроку из A, с позиции B до позиции C
UCASE(A)	Переводит A в верхний регистр

И наконец несколько просто полезных функций, которые тоже можно применить в select_expression.

CURDATE()	Текущая дата
DATABASE()	Имя текущей базы данных из которой выполняется выбор
FROM_DAYS()	Меняет день на DATE
NOW()	Текущее время в форматах YYYYMMDDHHMMSS или "YYYY-MM-DD HH:MM:SS". Формат зависит от того в каком контексте используется NOW(): числовом или строковом
PASSWORD()	Шифрует строку
PERIOD_ADD(P:N)	Добавить N месяцев к периоду P (в формате YYMM)
PERIOD_DIFF(A, B)	Возвращает месяцы между A и B. Обратите внимание, что PERIOD_DIFF работает только с датами в форме YYMM или YYYMM
TO_DAYS()	Меняет DATE (YYMMDD) на номер дня
UNIX_TIMESTAMP([date])	Возвращает метку времени unix, если вызвана без date (секунды, начиная с GMT 1970.01.01 00:00:00). При вызове со столбцом TIMESTAMP вернет TIMESTAMP. date может быть также строкой DATE, DATETIME или числом в формате YYMMDD (или YYYMMDD)
USER()	Возвращает логин текущего пользователя
WEEKDAY()	Возвращает день недели (0 = понедельник, 1 = вторник, ...)

Групповые функции в операторе select

Следующие функции могут быть использованы в предложении GROUP:

AVG()	Среднее для группы GROUP
SUM()	Сумма элементов GROUP
COUNT()	Число элементов в GROUP
MIN()	Минимальный элемент в GROUP
MAX()	Максимальный элемент в GROUP