Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Томский государственный университет систем управления и радиоэлектроники». (ТУСУР)

УТВЕРЖДАЮ

Заведующий кафедрой
«Управление инновациями»

_____ /*А.Ф.Уваров*
(подпись) *(ФИО)*
"_____" _____ 2012 г.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К ЛАБОРАТОРНЫМ ЗАНЯТИЯМ

по дисциплине

Базы данных

Составлена кафедрой «Управление инновациями»

Для студентов, обучающихся
по направлению подготовки 220600.62 «Инноватика»
по специальности 220601.65 «Управление инновациями»

Форма обучения очная

Составитель
к.т.н., Титков Антон Вячеславович
"_20_" _сентября_ 2012 г

Томск 2012 г.

СОДЕРЖАНИЕ

Введение

Изучение дисциплины «Базы данных» имеет важное значение в специальной подготовке студентов по направлению «Инноватика» и специальности «Управление инновациями». Цель данного пособия состоит в приобретение навыков работы с современными СУБД для создания и управления базами данных. Для полноценного понимания и усвоения материала необходимо предварительно изучить дисциплину "Информатика".

Для углубленного изучения и освоения материала целесообразно выполнение лабораторных работ, наряду с другими различными формами обучения студентов: тесты, задачи, упражнения, которые используются при проведении практических занятий в университете, выполнении контрольных и аудиторных работ, а также при самостоятельном изучении данных дисциплин.

Одним из наиболее интенсивных способов изучения дисциплины является самостоятельное выполнение лабораторных работ, на которых вырабатываются навыки взаимодействия с СУБД.

Предлагаемые лабораторные работы позволяют глубже освоить теоретические и практические вопросы, понять принципы проектирования баз данных и научиться управлять существующими решениями.

Лабораторная работа №1. Установка MySQL

Цель занятия: получить представление о СУБД MySQL, научиться работать в текстовом режиме.

Задание:

Установить на flash-накопитель СУБД MySQL.

Ход работы.

MySQL — свободно-распространяемая система управления реляционными базами данных, доступная для множества платформ. Изначально была разработана Майклом Видениусом и принадлежала специально созданной для развития этой СУБД компании MySQL AB. В 2008 году компания была приобретена Sun Microsystems, а с 2010 — стала принадлежать Oracle после того, как последняя поглотила Sun. Несмотря на все эти коммерческие метаморфозы, MySQL остается самой популярной СУБД в мире программного обеспечения с открытым кодом и одной из популярнейших в мире СУБД вообще.

MySQL имеет двойное лицензирование:

- GNU GPL (лицензия на свободное ПО);
- коммерческая лицензия, позволяющая использовать MySQL, не открывая исходный код своих продуктов, как того требует GPL (в случае использования/модификации кода самой СУБД MySQL).

У MySQL две основные версии:

- бесплатная — MySQL Community Server, именно она наиболее популярна: предоставляется хостинг-компаниями для веб-сайтов, входит в состав многочисленных Linux-дистрибутивов, интегрируется в программное обеспечение с открытым исходным кодом и так далее; для выполнения всех лабораторных работ будет использоваться именно эта версия;

- коммерческая — MySQL Enterprise, она основывается на бесплатной версии, но включает в себя ряд дополнительных инструментов для работы с базами данных, а также профессиональную техническую поддержку от компании-производителя (Oracle).

Кроме того, существует ряд отдельных продуктов, построенных на базе MySQL. Например, MySQL Cluster — самостоятельное решение, основанное на MySQL с хранилищем NDBCLUSTER и предназначенное для построения кластеров, а MySQL Embedded Database — редакция MySQL для использования в качестве встраиваемой СУБД.

На официальном сайте MySQL (http://www.mysql.com/) можно найти исчерпывающую информацию и документацию по этой СУБД.

Установка MySQL

Пакеты с MySQL Community Server для всех поддерживаемых программных платформ (среди них — GNU/Linux, Windows, Mac OS X, Solaris, FreeBSD, NetBSD, OpenBSD и многие другие) доступны для свободного скачивания на сайте http://dev.mysql.com/.

Для установки MySQL в среде операционной системы Windows загрузите соответствующий вашей системе файл (ZIP-архивы и самораспаковывающиеся инсталляторы в формате MSI доступны для 32- и 64-битных редакций Windows) и установите продукт согласно простым инструкциям инсталлятора. В качестве языковых настроек рекомендуется указать UTF-8.

Для установки MySQL в среде операционной системы GNU/Linux лучше всего воспользоваться средствами управления пакетами вашего дистрибутива. Например, в Ubuntu Linux для этого достаточно выполнить в терминале команду "sudo apt-get install mysql-client mysql-server" (для установки клиента и сервера MySQL), а в Fedora — "yum install mysql mysql-server".

Если же вы по каким-то причинам не хотите устанавливать MySQL средствами дистрибутива, можно воспользоваться готовыми бинарными пакетами для Linux, доступными на страницеhttp://dev.mysql.com/downloads/mysql/. Там же можно найти и архив с исходным кодом MySQL для самостоятельной компиляции СУБД. Но такой подход к инсталляции пакетов популярного и общедоступного (присутствующего в репозиториях многочисленных Linux-дистрибутивов) программного обеспечения **строго не рекомендуется**.

Во время установки вас попросят указать пароль администратора сервера баз данных. Если вы забудете введенный пароль после установки MySQL, его можно сменить. В Ubuntu Linux это можно сделать командой "sudo dpkg-reconfigure mysql-server-5.1" (здесь окончание "5.1" в названии "перенастраиваемого" пакета может изменяться в зависимости от версии MySQL, установленной в вашей системе).

Linux-пользователям для установки UTF-8 в качестве используемой по умолчанию в MySQL кодировки требуется добавить в секцию "[mysqld]" конфигурационного файла /etc/mysql/my.cnf следующие строки:

```
init-connect='SET NAMES utf8'
character-set-server=utf8
collation-server=utf8_general_ci
```

Если сервер MySQL уже был запущен, то после сохранения внесенных в конфигурационный файл изменений потребуется перезапустить сервер. Для этого

воспользуйтесь средствами используемой в вашем Linux-дистрибутиве системы управления службами. В Ubuntu Linux, например, достаточно выполнить в терминале команду "sudo /etc/init.d/mysql restart". Только после этого изменения вступят в силу, и UTF-8 станет кодировкой по умолчанию (как для данных в терминале, так и для хранения информации в БД).

Подключение к MySQL

Базовый интерфейс работы с СУБД MySQL — терминал. Существуют и другие (в том числе, графические) способы взаимодействия с этой СУБД, однако на начальных этапах знакомства с SQL-запросами **рекомендуется работать в классическом текстовом режиме.**

Поскольку MySQL обладает клиент-серверной архитектурой, после запуска *сервера* базы данных потребуется произвести подключение к нему с помощью специального *клиента*. Клиент вызывается командой "mysql", а в качестве аргументов принимает множество различных параметров. Ключевые параметры, которыми мы будем пользоваться при подключении:

- -u username или --user=username — имя пользователя в СУБД MySQL, с правами которого будут выполняться все команды. **Не стоит путать его с пользователем операционной системы** — эти учетные записи совершенно независимы. Для простоты мы будем работать из-под пользователя главного администратора СУБД "root".
- -p[password] или --password[=password] — пароль пользователя для подключения к СУБД. Квадратные скобки ("[" и "]") указывают на то, что **указывать сам пароль в качестве параметра необязательно** — в таком случае клиент попросит ввести пароль после исполнения команды (приглашением "Enter password:"), а пароль не будет передан в чистом виде в команде подключения (а значит — не попадет в историю команд интерпретатора и не будет отображаться в полном списке процессов операционной системы), что является распространенной техникой обеспечения элементарного уровня безопасности. Пароль пользователя "root" вы задаете на этапе установки сервера MySQL (см. "Установка MySQL").
- -h hostname или --host=hostname — IP-адрес или DNS-имя сервера, к которому будет производится подключение. Поскольку мы будем работать с сервером, находящимся на том же компьютере, что и клиент MySQL, используемым значением будет "localhost" или "127.0.0.1". Впрочем, по умолчанию клиент подключается именно к текущему компьютеру (т.е. "localhost"), поэтому указывать этот параметр в нашем случае вовсе не обязательно.
- dbname — еще один необязательный параметр, он указывается без дополнительных флагов и задает имя базы данных, к которой будет осуществляться подключение. Мы его использовать тоже не будем, поскольку создадим и выберем нужную базу данных уже после подключения к серверу.

Итак, в самом простом случае команда подключения к серверу MySQL будет выглядеть следующим образом:

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

После выполнения этой команды вас спросят пароль пользователя root в MySQL, а после его ввода (в случае правильного указания пароля) MySQL выведет приветствие и приглашение к началу работы на сервере. Заметьте, что **при вводе пароля в "Enter password:" вводимые с клавиатуры символы не отображаются на экране**, но при этом они на самом деле вводятся в поле — это сделано специально для того, чтобы

сторонние люди, находящиеся у вашего монитора, не могли узнать ни о том, какой пароль вы вводите, ни количество символов в нем.

Успешная процедура подключения будет выглядеть примерно следующим образом:

```
shurup@ubuntop:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 37
Server version: 5.1.37-1ubuntu5.1 (Ubuntu)

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

mysql>
```

Обратите внимание, что мы не указывали ни параметр host (по умолчанию это "127.0.0.1" или "localhost"), ни dbname (по умолчанию происходит подключение к серверу без выбора какой-либо определенной базы данных).

Лабораторная работа №2. Работа с базами данных и таблицами

Цель занятия: научиться создавать собственные базы данных и таблицы.

Задание:

Создайте базу данных и таблицу с несколькими атрибутами.

Ход работы.

После подключения к серверу терминал начинает принимать и выполнять как специальные внутренние команды (для вывода их списка и справки по ним наберите в терминале "help" и нажмите на Enter), так и запросы на языке SQL, используемом в MySQL.

Из курса лекций вы уже знаете, что операции, выполняемые над отношениями, условно делят на две группы:

- операции над множествами (объединение, пересечение, разность, деление, декартово произведение);
- операции над отношениями (проекция, соединение, выбор).

В реляционных СУБД для выполнения операций над отношениями используются две группы языков, в качестве математической базы для которых используются теоретические языки запросов, предложенные Эдгаром Коддом:

- реляционная алгебра;
- реляционное исчисление.

В первом случае (реляционная алгебра) операнды и результаты всех действий являются отношениями. Такие языки — процедурные, поскольку отношение, которое является результатом запроса к базе данных, вычисляется при последовательном выполнении операторов, применяемым к отношениям. Сами операторы состоят из операндов (отношений) и реляционных операций (их результатом тоже является отношение).

Языки реляционного исчисления — непроцедурные, их называют *описательными* или *декларативными*. Они позволяют составлять запросы с помощью предиката первого порядка, которому должны удовлетворять кортежи или

домены отношений. Таким образом, запрос к базе данных на таком языке содержит только информацию о желаемом результате. К языкам этой группы относится и SQL.

SQL (Structured Query Language, "язык структурированных запросов") — универсальный язык, применяемый для создания, модификации и управления данными в реляционных базах данных. В нем создается линейная последовательность операторов языка, которые выполняются СУБД. Операторы состоят из:

- имен операций и функций;
- имен таблиц и их столбцов;
- зарезервированных ключевых слов и специальных символов;
- логических, арифметических и строковых выражений.

Синтаксис SQL

Общий вид простого оператора в SQL:

```
ОПЕРАТОР параметры;
```

Если параметр не один, а несколько, то они перечисляются через запятую.

Все предложения на языке SQL оканчиваются точкой с запятой. Например:

```
USE `first_lab_data_base`;
SELECT `id`, `field1` FROM `mytable`;
```

Выражения в SQL не зависят от регистра, не требуют обязательного наличия кавычек при обозначении названий, дополнительные разделители (пробел, табуляция, переход на новую строку) игнорируются. Для обозначения названий баз данных, таблиц, атрибутов таблиц, то есть названий, связанных с объектами СУБД, могут использоваться кавычки типа «тупое ударение» («`»). Например:

```
`элемент_бд`
```

Для текстовых данных, вводимых пользователем в базу и не связанных с элементами СУБД (например, обычных строковых значений), используются обычные или двойные кавычки: 'текст', "большой текст". Чтобы записать в тексте кавычки так, что они не воспримутся СУБД как конец строки, кавычки и некоторые другие метасимволы экранируются символом обратной косой черты («\»). Например, чтобы с помощью двойных кавычек указать строку, которая уже содержит двойные кавычки в конце и начале выражения, нужно записывать ее так:

```
"\"заголовок\""
```

Существует общепринятый стиль «правильного» оформления выражений. Оно заключается в том, что при написании каких-либо выражений:

- после естественных разделителей выражений (например, запятых) ставится пробел;
- дополнительные разделители (пробелы, табы) не используются, если нет необходимости записать многостроковое выражение в удобном для чтения виде;
- системные обозначения (названия операторов, функций, ключевых слов и т.п.) пишутся заглавными буквами;

- при указании названий, связанных с объектами СУБД, обязательно используются кавычки в виде «тупого ударения».

Например:

```
INSERT INTO `news` (`id`, `post_date`) VALUES (42, '2008-06-01 04:13:15');
```

В данном методическом материале также **будут использоваться квадратные скобки для обозначения дополнительных (но необязательных) параметров**. Выражения, с параметром в квадратных скобках и без него, являются синтаксически правильными.

Например:

```
SELECT * FROM `table` [ WHERE условие ];
```

Такое выражение можно интерпретировать не только так:

```
SELECT * FROM `table` WHERE условие;
```

Но и как аналогичное выражение без необязательных параметров, т.е.:

```
SELECT * FROM `table`;
```

Типы данных в MySQL

В MySQL представлено множество типов данных, в соответствии с которыми хранятся и обрабатываются все данные в таблицах.

Числа

Числа разделяются на целые и дробные. Целые представлены следующими типами данных:

- TINYINT — 1 байт, т.е. принимает значения от -128 до 127 (в случае использования TINYINT UNSIGNED, т.е. без учета знака перед числом — 0..255);
- SMALLINT — 2 байта, -32768..32767 (0..65535);
- MEDIUMINT — 3 байта, -8388608..8388607 (0..2^24-1);
- INT — 4 байта, -2147483648..2147483647 (0..2^32-1);
- BIGINT — 8 байт, -2^32..2^32-1 (0..2^64).

Дробные числа представлены следующими типами данных:

- FLOAT (4 байта);
- DOUBLE (8 байт) — вдвое большая точность после запятой.

Строки

- CHAR — дополняет до заданной «ширины» (например, в случае использования CHAR(6) строка из пяти символов «hello» будет храниться как «hello », т.е. с пробелом на конце);
- VARCHAR — использует необходимый минимум памяти (в случае использования VARCHAR(6) строка из пяти символов «hello» будет храниться как «hello»);

- BLOB, TINYBLOB, MEDIUMBLOB, LONGBLOB — бинарные данные разных размеров;
- TEXT, TINYTEXT, MEDIUMTEXT, LONGTEXT — текстовые данные разных размеров;
- ENUM — одно из заданных значений (например, в ячейке типа ENUM(0,1,2) может храниться ноль, единица или двойка);
- SET — ноль или более заданных значений (в ячейке типа SET(0,1,2) может храниться любая комбинация из ноля, единицы и двойки — в том числе и пустое значение).

Другие типы

- булев: BOOL, BOOLEAN;
- уникальный автоматически увеличившийся идентификатор: SERIAL (== BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE);
- дата и время: DATETIME, DATE, TIMESTAMP, TIME, YEAR.

1. Команды для манипулирования базами данных

1.1. Создание базы данных:

```
CREATE DATABSE `db_name`;
```

Параметр команды создания баз данных — имя, выдаваемое создаваемой базе данных. Например, для создания базы данных под названием "my_database" нужно ввести команду:

```
CREATE DATABSE `my_database`;
```

Несмотря на то, что современная версия MySQL позволяет создавать БД с кириллическими и специальными символами в названии, принято использовать латинские буквы, цифры и знаки подчеркивания («_»).

1.2. При одновременной работе в нескольких базах данных в командах нужно уточнять, с данными какой БД вы работаете. Для этого используется разделитель точка — «.». Так, чтобы обратиться к атрибуту "attribute" таблицы "table", находящейся в базе данных "database1", нужно использовать запись:

```
`databse1`.`table`.`attribute`
```

Если же вам понадобится обратиться к аналогичному атрибуту такой же таблицы, находящейся в БД "database2", запись станет такой:

```
`databse2`.`table`.`attribute`
```

Для того, чтобы вводимые команды применялись к конкретной базе данных по умолчанию, можно воспользоваться командой USE и ввести название базы данных, с которой мы будем в дальнейшем работать:

```
USE `my_databse`;
```

После выполнения команды USE следующие записи будут эквивалентны:

```
`my_database`. `table`
`table1`
```

1.3. Удаление существующей базы данных выполняется командой DROP DATABASE, которая в качестве единственного аргумента принимает название удаляемой базы данных. Например, чтобы удалить созданную вами в начале работы БД "my_database", нужно выполнить:

```
DROP DATABASE `my_database`;
```

После успешного удаления вы можете заново создать ее. **Учтите, что если бы в вашей базе данных были таблицы и данные, вся эта информация была бы утеряна навсегда.**

1.4. Для просмотра информации о базах данных, их таблиц, а также привилегий текущего пользователя, используется команда SHOW.

1.4.1 Увидеть список всех доступных пользователю баз данных можно с помощью команды:

```
SHOW DATABASES;
```

1.4.2 Увидеть список всех таблиц в используемой базе данных можно с помощью команды:

```
SHOW TABLES;
```

1.4.3 Для вывода информации о таблице "table_name" используйте команду:

```
SHOW CREATE TABLE `table_name`;
```

Обратите внимание, что результатом выполнения команды будет команда создания таблицы с учетом всех изменений, произведенных над таблицей в процессе работы с базой данных. Более подробную информацию о создании таблиц в MySQL см. ниже в подразделе «Команды для работы с таблицами БД».

1.4.4 Увидеть список всех прав текущего пользователя СУБД можно с помощью команды:

```
SHOW GRANTS;
```

2. Команды для работы с таблицами БД

2.1. Для **создания таблицы** используется команда CREATE TABLE. В качестве аргумента ей передается название новой таблице и перечисление всех атрибутов таблицы и их описаний (типы данных, значения по умолчанию, ограничения на применяемые значения и т.п.) — все перечисление берется в круглые скобки, а разделителем между атрибутами служит запятая. В общем виде команда выглядит так:

```
CREATE TABLE `table_name` (
 `название_первого_поля` его_тип [параметры],
 `название_второго_поля` его_тип [параметры],
 ...
 `название_последнего_поля` его_тип [параметры]
);
```

Обратите внимание, что после последнего атрибута запятая не требуется. Пример создания таблицы "news" с тремя атрибутами (идентификатор новости, время публикации, текст новости):

```
CREATE TABLE `news` (
  `id` MEDIUMINT(8) UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
  `posted` TIMESTAMP NOT NULL,
  `content` TEXT
);
```

2.2. Для **удаления таблицы** используется команда DROP TABLE с названием таблицы в качестве единственного аргумента. Например, для удаления созданной таблицы "news" команда будет выглядеть так:

```
DROP TABLE `news`;
```

2.3. Для **изменения таблиц** используется команда ALTER TABLE. Вид производимого изменения определятся последующими дополнительными командами:

2.3.1. **Переименование** таблицы осуществляется с помощью подкоманды RENAME. Например, чтобы переименовать таблицу "news" в "news_new", нужно выполнить следующую команду:

```
ALTER TABLE `news` RENAME TO `news_new`;
```

2.3.2. Для **добавления нового атрибута в таблицу** потребуется подкоманда ADD COLUMN, для которой нужно ввести название нового атрибута и указать его тип. Например, добавление к таблице "news" нового атрибута "author" (имя автора) будет выглядеть следующим образом:

```
ALTER TABLE `news` ADD COLUMN `author` VARCHAR(42);
```

Кроме того, можно задать положение добавляемого поля. Для этого в конец команды добавляется инструкция, указывающая, после какого столбца будет добавлено новое поле. Например, команда добавления атрибута "author" в таблицу "news" с тем, чтобы "author" стал вторым полем, выглядит следующим образом:

```
ALTER TABLE `news` ADD COLUMN `author` VARCHAR(42) AFTER `id`;
```

Чтобы поле стало первым в таблице, нужно заменить конструкцию с "AFTER ..." на ключевое слово "FIRST". Команда добавления атрибута "author" в таблицу "news" в качестве первого поля будет выглядеть так:

```
ALTER TABLE `news` ADD COLUMN `author` VARCHAR(42) FIRST;
```

2.3.3. Для **изменения типа атрибута таблицы** служит подкоманда MODIFY, для которой нужно указать таблицу, название атрибута и заново перечислить все требуемые для него параметры. Например, чтобы изменить тип атрибута "author" таблицы "news" на CHAR(42), нужно выполнить следующую команду:

```
ALTER TABLE `news` MODIFY COLUMN `author` CHAR(42);
```

2.3.4. Для **удаления атрибута из таблицы** служит подкоманда DROP COLUMN. Например, команда удаления атрибута "author" из таблицы "news" выглядит так:

```
ALTER TABLE `news` DROP COLUMN `author`;
```

3. Команды для работы с данными таблиц баз данных

Все команды из двух предыдущих подразделов («Команды для манипулирования базами данных» и «Команды для работы с таблицами БД») относятся к языку DDL (Data Definition Language), являющемуся частью языка SQL, которая обеспечивает работу со схемой БД. Другая, не менее важная, часть SQL — язык DML (Data Manipulation Language), позволяющий манипулировать самими данными в таблицах баз данных. К нему относятся команды, речь о которых пойдет уже в этом подразделе.

3.1. Для **добавления строк в таблицу** используется команда INSERT. В качестве аргументов ей передается название таблицы и набор всех значений для одной из строк или набор из названий атрибутов и соответствующих им значений, подразумевая, что значения остальных атрибутов будут заполнены автоматически.

Пример добавления строки в таблицу "news" с 4 атрибутами ("id", "posted", "content", "author") с указанием всех значений:

```
INSERT INTO `news` VALUES (1, NOW(), "Текст новости", "Редактор");
```

Использованное в запросе выражение "NOW()" — это обращение к встроенной в MySQL функции, которая возвращает текущее время, Таким образом, в качестве значения "posted" для добавляемой строки запишется текущее время.

Пример добавления строки в таблицу "news" с 4 атрибутами ("id", "posted", "content", "author") с указанием только некоторых значений:

```
INSERT INTO `news` (`posted`, `content`) VALUES (NOW(), "Текст новости");
```

В данном случае для добавляемой строки мы задали только значения атрибутов "posted" и "content". В поле "id" автоматически запишется число, которое будет на единицу больше последнего значения "id" в таблице (или 1, если добавляемая строка — первая), а в поле "author" для этой строки запишется значение NULL — отметка об отсутствии значения (**не путайте ее с нулевым значением**).

3.2. Для **изменения данных в строке таблицы** используется команда UPDATE. В качестве аргументов ей передаются название таблицы, названия атрибутов, значения которых будут изменены, и новые значения этих атрибутов.

Например, для того, чтобы изменить значение поля "author" во всех строках таблицы "news" на "Пользователь", а значение поля "content" — на "Пустой текст", нужно выполнить следующую команду:

```
UPDATE `news` SET `author` = "Пользователь", `content` = "Пустой текст";
```

В данном случае будут изменены все строки таблицы, но зачастую нужно изменять значения только в определенных строках. Для этого предусмотрена возможность установления условия отбора тех строк (WHERE), для которых будут проведены изменения.

Например, чтобы теперь изменить значение поля "content" на "Новый текст" только для строки с "id", равным 1, в таблице "news", нужно выполнить следующую команду:

```
UPDATE `news` SET `content` = "Новый текст" WHERE id = 1;
```

Правила формирования условий запросов подробно описаны в следующем подразделе — «Команда выборки».

3.3. Для **удаления данных из таблицы** используется команда DELETE. В качестве аргумента ей передается только название таблицы. Кроме того, можно ограничить список удаляемых строк, задав условие с помощью WHERE (по аналогии с тем, как это делалось в команде UPDATE) — тогда СУБД выберет строки, соответствующие условию, и удалит из таблицы только их. Без указания условия будут удалены все строки из таблицы.

Например, чтобы удалить из таблицы "news" строку с "id", равным 1, нужно выполнить следующую команду:

```
DELETE FROM `news` WHERE id = 1;
```

Лабораторная работа №3. Команды выборки данных из таблицы

Цель занятия: усвоить синтаксис, основные возможности языка SQL и основные принципы построения запросов на нем.

Задание:

Написать несложные SQL-запросы для исполнения в реляционной СУБД на примере MySQL.

Ход работы.

Для получения значений атрибутов таблиц БД используется команда SELECT. Общий вид команды представляется следующим образом:

```
SELECT `my_field1`, `my_field2`, ..., `my_fieldN`
FROM `my_table`
WHERE условие;
```

Здесь:

- my_field1, my_field2 и т.д. — это перечисление названий атрибутов, значения которых мы "выбираем", т.е. в результирующей таблице будут выведены только значения указанных атрибутов. В случае, если требуется вывод значений всех атрибутов

результирующей таблицы, для упрощения записи запроса используется символ звездочки («*»).

- my_table — это название таблицы, из которой будет сделана выборка.
- условие WHERE может иметь сложную структуру или отсутствовать.

Например, для выборки всех значений всех атрибутов из таблицы "news" достаточно ввести следующую команду:

```
SELECT * FROM `news`;
```

При выполнении запроса в оперативной памяти создается виртуальная таблица, состоящая из всех данных, удовлетворяющих условию отбора, а исходная таблица при этом никак не изменяется.

2. Условия выборки

Для решения некоторых задач условие отбора (WHERE) может иметь сложную структуру. Для построения таких условий используют логические операторы AND, OR, NOT и некоторые другие возможности языка SQL. Для группировки условий используются разделительные скобки («(» и «)»).

1. Для сравнения выражений предусмотрены: равенство («=»); больше или равно («>=»), меньше или равно («<=»), не равно («<>» или «!=»). Например, выборка значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" меньше 42, осуществляется следующим образом:

```
SELECT `content` FROM `news` WHERE `id` < 42;
```

2. Логическое умножение (И) записывается как AND и используется, когда требуется одновременное выполнение двух и более условий. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" больше 21 и меньше 42:

```
SELECT `content` FROM `news` WHERE `id` > 21 AND `id` < 42;
```

3. Логическое сложение (ИЛИ) записывается как OR и используется, когда требуется, чтобы выполнялось хотя бы одно из нескольких условий. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" больше 21 или меньше 10:

```
SELECT `content` FROM `news` WHERE `id` > 21 OR `id` < 10;
```

Логическое сложение имеет меньший приоритет чем логическое умножение, поэтому для корректной записи логических формул может потребоваться использование разделяющих скобок. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" либо больше 21 и меньше 42, либо больше 84:

```
SELECT `content` FROM `news` WHERE ( `id` > 21 AND `id` < 42 ) OR `id`
> 84;
```

4. Логическое отрицание (НЕ) записывается как NOT и используется для инвертирования последующего условия. Например, выборку значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" меньше 21 или больше 42, можно осуществить так:

```
SELECT `content` FROM `news` WHERE NOT ( `id` >= 21 AND `id` <= 42 );
```

5. Если у атрибута отсутствует значение, то оно записывается как NULL *(NULL — символ отсутствия значения, что не путать с пустым значением: пустое значение существует, а NULL указывает на его отсутствие)*. Для составления условий на выборку подобных отсутствующих значений предусмотрено специальное выражение IS NULL. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "author" отсутствует:

```
SELECT `content` FROM `news` WHERE `author` IS NULL;
```

6. Для указания принадлежности значения атрибута какому-либо интервалу предусмотрено выражение BEETWEEN .. AND. Выражение "BETWEEN a AND b". Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" больше 21 и меньше 42:

```
SELECT `content` FROM `news` WHERE `id` BETWEEN 21 AND 42;
```

7. Для указания принадлежности значения атрибута какому-либо множеству предусмотрено выражение IN. Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "id" принимает значения 21, 42, 84 или 168:

```
SELECT `content` FROM `news` WHERE `id` IN (21, 42, 84, 168);
```

8. Для поиска строковых значений, содержащих заданную строку по шаблону, предусмотрена выражение LIKE. В качестве аргумента оператору LIKE передается шаблон в виде строки, в которой помимо текста могут содержаться метасимволы «_» (обозначает любой одиночный символ) и «%» (набор любых символов любой длины). Пример выборки значений атрибута "content" из таблицы "news", в строках которой значение атрибута "author" соответствует шаблону «_user%» (т.е. строка, которая строится как любой символ + user + любая последовательность символов):

```
SELECT `content` FROM `news` WHERE `author` LIKE "_user%";
```

Такому шаблону будут удовлетворять значения атрибута вроде «1user», «1user222», «xuser42» и т.д.

3. Сортировка и ограничение результатов

Для **ограничения количества результирующих строк**, удовлетворяющих условию отбора, используется выражение LIMIT, которое дописывается в конец запроса с максимальным числом строк для вывода в качестве простейшего аргумента.

Пример для выборки информации о 5 первых новостях из таблицы "news" со значениями атрибута "id" больше 5:

```
SELECT * FROM `news` WHERE `id` > 5 LIMIT 5;
```

Для **сортировки результата отбора** предусмотрен оператор ORDER BY. Он позволяет сортировать строки в результирующей таблице в соответствии со значениями выбранных атрибутов (если атрибутов несколько, то приоритет сортировки убывает при перечислении атрибутов слева направо). Поддерживается два типа сортировки: по возрастанию (ASC, этот режим используется по умолчанию) и по убыванию (DESC).

Пример для выборки информации о 10 первых новостях из таблицы "news" со значениями атрибута "id" больше 5, отсортированной сначала по авторам ("author") в алфавитном порядке, а затем — по идентификаторам ("id") в обратном порядке:

```
SELECT * FROM `news` WHERE `id` > 5 ORDER BY `author` ASC, `id` DESC LIMIT
10;
```

4. Числовые операции (агрегирующие функции)

1. Для получения среднего арифметического значения атрибута по всем результирующим строкам предусмотрена функция AVG(). В качестве единственного аргумента ей передается название атрибута, значения которого будут учитываться. Пример выборки для нахождения среднего значения идентификатора ("id") новости:

```
SELECT AVG(`id`) FROM `news`;
```

2. Для поиска минимального и максимального значений атрибута среди всех результирующих строк есть функции MIN() и MAX() соответственно. В качестве единственного аргумента им передается название атрибута, значения которого будут учитываться. Пример выборки для нахождения минимального и максимального значений идентификатора ("id") новости:

```
SELECT MIN(`id`), MAX(`id`) FROM `news`;
```

3. Для подсчета суммы числовых значений атрибута среди всех результирующих строк предусмотрена функция SUM(). В качестве единственного аргумента ей передается название атрибута, значения которого будут учитываться. Пример выборки для нахождения суммы значений идентификаторов ("id") всех новостей:

```
SELECT SUM(`id`) FROM `news`;
```

4. Для подсчета количества строк выборки предусмотрена функция COUNT(). В качестве единственного аргумента ей передается название атрибута, значения которого будут учитываться. Кроме того, поскольку обычно не важно, по какому атрибуту считать число результирующих строк, вместо названия атрибута в качестве аргумента может использоваться метасимвол звездочки («*»). Пример выборки для нахождения количества строк в таблице "news":

```
SELECT * FROM `news` GROUP BY `author`;
```

Дополнительный параметр DISTINCT, указанный перед названием атрибута, позволяет вычесть из результата все вхождения по повторяющимся значениям выбранного поля. Пример выборки для нахождения количества новостей, написанных разными авторами:

```
SELECT COUNT(DISTINCT `author`) FROM `news`;
```

5. Группировка результатов

Для разбиения результатов выборки по группам используется оператор GROUP BY. Разбиение строк по группам бывает необходимо для проведения каких-либо операций не над всеми строками по отдельности, а над группами, отобранными по какому-либо атрибуту и условию.

Группировка производится по указываемому атрибуту (или набору атрибутов), и строки распределяются по группам в соответствии со значением атрибута в этой строке (каждую группу образует набор строк с одним значением атрибута, по которому производится группировка).

Пример выборки значений всех атрибутов таблицы "news" с группировкой по атрибуту "author":

```
SELECT * FROM `news` GROUP BY `author`;
```

С помощью группировки можно, например, посчитать количество новостей, добавленных каждым автором:

```
SELECT `author`, COUNT(*) FROM `news` GROUP BY `author`;
```

Для добавления условий на результат группировки в конструкцию GROUP BY добавляют выражение HAVING, работающее по аналогии с WHERE, но с группами строк.

Например, к прошлому запросу с помощью HAVING можно добавить условие, по которому будут выводиться только те авторы, которые добавили более 3 новостей:

```
SELECT `author`, COUNT(*) AS `cnt` FROM `news` GROUP BY `author` HAVING
`cnt` > 3;
```

Лабораторная работа №4. Работа с несколькими таблицами

Цель занятия: научиться создавать первичные и внешние ключи, делать запросы сразу к нескольким таблицам.

Задание:

Создать две таблицы, связанные внешним ключом и сделать выборки, используя запросы сразу к нескольким таблицам.

Ход работы.

1. Ключи

Первичный ключ

Пример создания:

```
CREATE TABLE `t1` (
 `id` BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
 ...
) TYPE=InnoDB;
```

Внешний ключ

Пример создания:

```
CREATE TABLE `laptops` (
 `id` SERIAL PRIMARY KEY,
 `manufacturer_id` BIGINT UNSIGNED NOT NULL,
 CONSTRAINT `made_by` FOREIGN KEY(`manufacturer_id`) REFERENCES
`manufacturers`(`id`)  ON DELETE CASCADE
) TYPE=InnoDB;
```

2. Соединения таблиц

Способы образования новых таблиц из исходных:

1. Декартово произведение — построение новой таблицы из двух или более исходных путем включения в нее строк, образованных всеми возможными вариантами конкатенации (слияния) строк исходных таблиц. Количество строк новой таблицы определяется как произведение количеств строк всех исходных таблиц.
2. Ограничение — построение новой таблицы из исходной, путем включения в нее тех строк исходной таблицы, которые отвечают некоторому критерию в виде логического условия (ограничения).
3. Проекция — построение новой таблицы из исходной, путем включения в нее избранных столбцов исходной таблицы.
4. Объединение — построение новой таблицы из двух или более исходных, путем включения в нее всех строк исходных таблиц (при условии, что они подобны).

В чистом виде встречается только декартово произведение (операнд CROSS JOIN). Остальные операнды соединения таблиц происходят в несколько этапов, используя разные вышеперечисленные операции над таблицами.

Выборка из нескольких таблиц

Выборка одновременно из нескольких таблиц будет осуществляться в два этапа: на первом будет найдено декартово произведение опрашиваемых таблиц; на втором произойдет непосредственно выборка из получившейся таблицы.

Декартово произведение — построение новой таблицы из двух или более исходных таблиц, путем включения в нее строк, образованных всеми возможными вариантами конкатенации (слияния) строк исходных таблиц.

Таблицы перечисляются через запятые. Условие отбора WHERE имеет точно такие же параметры, как и для выборки одиночной таблицы

```
SELECT `field1`, `field2`, ..., `fieldN`
FROM `table1`, `table2`, ..., `tableN`
WHERE условие ;
```

CROSS JOIN и INNER JOIN

В СУБД MySQL так реализуют операцию декартова произведения таблиц:

```
SELECT * FROM `join1` CROSS JOIN `join2`;
SELECT * FROM `join2` INNER JOIN `join1`;
```

Введя дополнительные команды, можно:

- добавить условие соединения по полям таблиц:

```
SELECT * FROM `join1`  INNER JOIN `join2` ON `join1`.`surname` =
`join2`.`text`;
```

- произвести декартово произведение по полям таблиц с одинаковыми названиями:

```
SELECT * FROM `join1` INNER JOIN `join2` USING(`id_join2`);
SELECT * FROM `join1` INNER JOIN `join2` USING(`name`, `id_join2`);
```

FULL (OUTER) JOIN

Большинство соединений связывают строки одной таблицы со строками другой, но в некоторых случаях вам может понадобиться включать в результат строки, не имеющие связей. Таким образом, мы сможем увидеть те строки, в которых нет связи между соединяемыми таблицами. Внешние соединения также включают в результат строки, не имеющие связанных с ними строк.

```
SELECT * FROM `join1` FULL JOIN `join2`;
```

LEFT (OUTER) JOIN и RIGHT (OUTER) JOIN

При использовании синтаксиса OUTER JOIN вы должны использовать ключевое слово RIGHT или LEFT, чтобы указать таблицу, все строки которой будут включены в результат.

Следующие записи эквивалентны:

```
SELECT * FROM `join1` LEFT JOIN `join2` ON join1.id_join2 = join2.id_join2;
SELECT * FROM `join2` RIGHT JOIN `join1` ON join1.id_join2 = join2.id_join2;
```

В обоих случаях будут показаны все строки связей с `join1`, но в отличие от FULL (OUTER) JOIN не будут выведены несвязные строки таблицы `join2`.

NATURAL JOIN

Операция естественного соединения выражается через операции переименования, соединения общего вида и проекции. Для упрощения записи используется операнд NATURAL JOIN:

```
SELECT * FROM `join1` NATURAL JOIN `join2`;
```

3. Объединения запросов

UNION

UNION используется для объединения результатов работы нескольких команд SELECT в один набор результатов:

```
(SELECT join1.name, join1.surname AS text FROM join1) UNION (SELECT
join2.name, join2.text FROM join2);
```

Круглые скобки необходимы, когда требуется применять оператор ORDER BY и другие. Кроме того, их желательно использовать для визуального разделения запросов.

Конструкция "join1.surname AS text" означает, что для атрибута "join1.surname" присвоен псевдоним "text" с целью исключения совпадения имен в результате запроса и для упрощения работы в СУБД.

UNION ALL и UNION DISTINCT

Если не используется ключевое слово ALL для UNION, все возвращенные строки будут уникальными, так как по умолчанию подразумевается DISTINCT для всего результирующего набора данных. Если указать ключевое слово ALL, то результат будет содержать все найденные строки из всех примененных команд SELECT:

```
(SELECT `join1`.`name`, `join1`.`surname` AS `text` FROM `join1`) UNION ALL
(SELECT `join2`.`name`, `join2`.`text` FROM `join2`);
```

Сравните результат этого запроса с предыдущим.

4. Подзапросы

При работе с базой данных может возникнуть потребность в запросе, который зависел бы от результата другого запроса. Подзапрос - это запрос, результат которого используется в условии другого запроса.

```
SELECT * FROM `join1` WHERE `join1`.`id_join2` =
  (SELECT `join2`.`id_join2` FROM `join2` WHERE `join2`.`id_join2` > 1);
```

На подзапросы можно налагать дополнительные ограничения, такие как:

1) Оператор EXISTS. Используется, чтобы указать предикату, производит ли подзапрос вывод. Возвращает булево значение:

```
SELECT * FROM `join1`
  WHERE EXISTS `join1`.`id_join2` = (SELECT `join2`.`id_join2` FROM `join2`
WHERE `join2`.`id_join2` > 1);
```

2) Операторы SOME/ANY (взаимозаменяемые — различие в терминологии состоит в том, чтобы позволить людям использовать тот термин, который наиболее однозначен). Оператор ANY берет все значения выведенные подзапросом, причем такого же типа, как и те, которые сравниваются в основном предикате. В этом его отличие от EXISTS, который просто определяет, производит ли подзапрос результаты, и фактически не использует эти результаты.

```
SELECT * FROM `join1`
  WHERE `join1`.`id_join2` = ANY (SELECT `join2`.`id_join2` FROM `join2`);
```

3) Оператор ALL. Предикат является верным, если каждое значение выбранное подзапросом удовлетворяет условию в предикате внешнего запроса:

```
SELECT * FROM `join1`
```

```
WHERE `join1`.`id_join2` = ALL (SELECT `join2`.`id_join2` FROM `join2`);
```

Лабораторная работа №5. Управление пользователями и оптимизация запросов

Цель занятия: научиться создавать пользователей, назначать им права, уметь создавать представления и индексы.

Задание:

Создать двух пользователей с правами только на редактирование и администратора БД. Сконструировать представление, создать индекс по одному из атрибутов таблицы.

Ход работы.

1. Пользователи и их привилегии

Пользователи делятся на три вида, по способу доступа к СУБД:

1. анонимные,
2. локальные localuser@localhost,
3. глобальные(внешние) globaluser@globalhost.

Также различны и права пользователей:

1. select – право на использование операнда выборки над таблицей. Право на просмотр таблиц
2. insert, update, delete – соответственно внесение данных, изменение и удаление. Право на изменение содержимого таблиц
3. Административные права — право давать другим пользователям, отменять, изменять права и так далее.
4. Специальные права.

1.1. Создание/переименование/удаление пользователей

1. При создании пользователя, нужно указать имя пользователя, его способ доступа (с локальной машины или удаленно) и выдать ему пароль:

```
CREATE USER 'user'@'host' IDENTIFIED BY 'password';
```

2. Переименование пользователя

```
RENAME USER 'user' TO 'user2';
```

3. Удаление пользователя

```
DROP USER 'user'@'host';
```

1.2. Смена пароля пользователя

В команде указывается пользователь и новый пароль, который будет шифроваться операндом PASSWORD. Для смены пароля пользователя, нужно обладать соответствующим административным правом.

```
SET PASSWORD FOR 'user'@'host' = PASSWORD('newpassword');
```

1.3. Команды GRANT, REVOKE

GRANT — команда выдачи привилегий. Параметры операнда: Объект — базы данных и их таблицы. Если привилегии выдаются под конкретную таблицу, обязательно указывать к какой базе данных относится данная таблица, используя разделитель ".". Так же можно указать конкретные поля таблицы, на которые будут распространяться выдаваемые права. Субъект — пользователь базы данных. Важно указать тип доступа пользователя (с локальной или удаленной машины), чтобы повысить безопасность. Метасимвол '%' означает все типы пользователей.

```
GRANT
    PRIVILEGES(field1, fieldn)
   ON `database`.`table`
   TO 'user'@'%';
GRANT
    SELECT (`name`,`city`),
    INSERT (`name`)
   ON `test`.`girls`
   TO 'shurup'@'%';
```

Снятие привилегий происходит аналогично выдаче:

```
REVOKE priv_type [(cols)]
   ON [object_type]
   `db_name`.`table_name`
   FROM 'user'@'host';
```

2. Представления

Представление — запрос на выборку, сохраненный в базе данных под каким-то названием, виртуальная таблица. Данные представления динамически рассчитываются по запросу из данных реальных таблиц, но структура (поля) результата запроса не меняются при изменении исходных таблиц.

Плюсы использования виртуальных таблиц:

1. Повышение скорости работы. Когда прикладной программе требуется таблица с определённым набором данных, она делает простейший запрос из подготовленного представления. Поскольку SQL-запрос, выбирающий данные представления, зафиксирован на момент его создания, СУБД получает возможность применить к этому запросу оптимизацию или предварительную компиляцию, что положительно сказывается на скорости обращения к представлению по сравнению с прямым выполнением того же запроса из прикладной программы.
2. Независимая модификация прикладной программы и схемы хранения данных.
3. Повышение безопасности. За счет предоставления пользователю только тех данных, на которые он имеет права — от него скрыта реальная структура таблиц базы данных.

Может возникнуть такая ситуация, что в исходных таблицах есть обязательные и не пустые (NOT NULL) поля, а в представлении их нет. Тогда операция добавления строки данных не может быть выполнена за одно действие. Во избежание потерь данных подобных действий следует избегать.

Операнды для работы с представлениями

1. Создание:

```
CREATE VIEW `name` [ FIELDS ] AS { запрос };
```

2. Удаление:

```
DROP VIEW `name`;
```

3. Изменение:

```
ALTER VIEW `name` [ FIELDS ] AS { новый запрос }
```

3. Индексы

Индексы применяются для быстрого поиска строк с указанным значением одного столбца. Без индекса чтение таблицы осуществляется по всей таблице начиная с первой записи, пока не будут найдены соответствующие строки. Если же таблица содержит индекс по рассматриваемым столбцам, MySQL может быстро определить позицию для поиска в середине файла данных — без просмотра всех данных. Индексы хранятся в отдельных файлах в виде бинарных деревьев (B-tree). Поэтому СУБД требуется время для поддержания актуальности индексов при изменении, добавлении и удалении проиндексированных данных.

Что следует учитывать при создании индексов

1. Если использование индекса требует от MySQL прохода более чем по 30% строк в данной таблице, в таком случае просмотр всей таблицы, по всей видимости, окажется намного быстрее, так как потребуется выполнить меньше операций поиска. Поэтому индексы не используют для малых таблиц. Индексы дают прирост производительности при большом разбросе значений индексированных данных.
2. Желательно индексировать только наиболее часто опрашиваемые поля, т.к. время для поддержания актуальности индексов таблицы сильно зависит от их количества и может превысить выигрыш при поиске.
3. Особенность строения индекса как бинарного дерева приводит к тому, что индекс с известным началом ("index%") работает много лучше чем, если известна его середина ("%index" или "%index%").
4. При создании многостолбцовых индексов следует учитывать порядок создания индексов. Например, создан индекс по столбцам index1-index2-index3 (обратите внимание на их порядок). Из-за структуры индекса в виде B-дерева (B-tree) он будет эффективен при запросах, в которых участвуют следующие индексы (именно в таком порядке): index1-index2-index3, index1-index2, index1. В остальных случаях использования проиндексированных столбцов ("index1", "index2", "index3") индекс не повлияет на производительность.

Операнды работы с индексами

1. Создание индекса:

```
CREATE INDEX `indexName` ON `table1` (`fieldA`, `fieldB`, ...);
```

2. Удаление:

```
DROP INDEX `name` ON `table1`;
```

3. Использование индекса. Эффективность использования индекса зависит от задачи:

- эффективен при соединении таблиц, т.к. в проиндексированных полях упрощается поиск данных для сопоставления, а значит — соединение работает в целом быстрее;
- эффективен при поиске по значению индекса;
- не годится для выборки всей таблицы.

```
SELECT
  `index_a`
FROM
  `table1` LEFT JOIN `table2`
  ON `table1`.`index_b` = `table2`.`index_c`
WHERE
  `index_d` = expr;
```

Полнотекстовый поиск

Полнотекстовый поиск — один из вариантов использования индексов, поддерживаемый только в движке MyISAM для типов данных CHAR, VARCHAR, TEXT. Используется для поиска подстрок в строках таблиц:

```
CREATE TABLE `articles` (
  `id` INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
  `title` VARCHAR(200),
  `body` TEXT,
  FULLTEXT (`title`, `body`)
);

INSERT INTO `articles` (`title`, `body`) VALUES
  ('MySQL Tutorial','DBMS stands for DataBase ...'),
  ('How To Use MySQL Well','After you went through a ...'),
  ('Optimizing MySQL','In this tutorial we will show ...'),
  ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
  ('MySQL vs. YourSQL','In the following database comparison ...'),
  ('MySQL Security','When configured properly, MySQL ...');

SELECT * FROM `articles` WHERE MATCH (`title`, `body`) AGAINST ('database');
+----+-------------------+------------------------------------------+
| id | title             | body                                     |
+----+-------------------+------------------------------------------+
|  5 | MySQL vs. YourSQL | IN the following DATABASE comparison ... |
|  1 | MySQL Tutorial    | DBMS stands FOR DATABASE ...             |
+----+-------------------+------------------------------------------+

SELECT `id`, `body`, MATCH (`title`, `body`) AGAINST
  ('Security implications of running MySQL as root') AS `score`
  FROM `articles` WHERE MATCH (`title`, `body`) AGAINST
  ('Security implications of running MySQL as root');
```

```
+----+-------------------------------------+-----------------+
| id | body                                | score           |
+----+-------------------------------------+-----------------+
|  4 | 1. Never run mysqld AS root. 2. ... | 1.5219271183014 |
|  6 | When configured properly, MySQL ... | 1.3114095926285 |
+----+-------------------------------------+-----------------+
```

Долгие запросы и их анализ

Довольно часто возникают проблемы с выполнением тех или иных сложных запросов, выливающиеся в возрастание их времени выполнения, при этом не всегда понятно на каком именно запросе возникает возрастание времени выполнения. Для обнаружения подобных проблем можно использовать штатное средство MySQL -- журналы долгих операций. В конфигурационный файл my.cnf следует внести следующие коррективы:

```
log-slow-queries=/path/TO/slow_queries.log
long_query_time=10
//где long_query_time -- количество секунд выполнения запроса, после которых
он попадет в журнал медленных запросов
```

Либо запустить сервер MySQL с параметром:

```
-log-slow-queries[=/tmp/slow_queries.log]
```

Подобным образом за счет параметра

```
-log-queries-not-using-indexes
```

можно включить в журнал сведения о запросах, не использующих индексы.

Пример фрагмента журнала медленных запросов:

```
/usr/sbin/mysqld, Version: 5.0.84-log (Gentoo Linux mysql-5.0.84-r1).
started WITH:
Tcp port: 3306  Unix socket: /var/run/mysqld/mysqld.sock
Time                 Id Command    Argument
# Time: 091104 15:39:23
# User@Host: andrey[andrey] @ localhost []
# Thread_id: 2  Schema: laba_index
# Query_time: 82.416012  Lock_time: 0.342861  Rows_sent: 0  Rows_examined: 0
Rows_affected: 1050024  Rows_read: 1050025
USE `laba_index`;
UPDATE `example1` SET `Delta`= (`Tx`-`Rx`)*10*rand();
```

Оператор EXPLAIN

```
EXPLAIN `tbl_name` -- эквивалент для DESCRIBE `tbl_name`.

EXPLAIN SELECT select_options -- анализ оптимизатором MySQL выполнения
запроса.

EXPLAIN SELECT * FROM `example1` IGNORE INDEX
(`comboTR`,`comboRT`,`comboRTD`,`Txindex`) WHERE `Rx`<500 AND `Tx`<500;
+----+-------------+----------+-------+---------------+---------+---------+-
-----+------+-------------+
```

```
| id | select_type | TABLE     | type   | possible_keys | KEY      | key_len |
ref   | rows | Extra       |
+----+-------------+----------+-------+---------------+---------+---------+-
-----+------+-------------+
|  1 | SIMPLE      | example1 | range | RXindex       | RXindex | 5       |
NULL | 4502 | USING WHERE |
+----+-------------+----------+-------+---------------+---------+---------+-
-----+------+-------------+


EXPLAIN SELECT * FROM `example1` USE INDEX
(`comboTR`,`comboRT`,`comboRTD`,`Txindex`) WHERE `Rx`<500 AND `Tx`<500;
+----+-------------+----------+-------+-------------------------------------+--
-------+---------+------+------+-------------------------+
| id | select_type | TABLE     | type   | possible_keys                       |
KEY      | key_len | ref   | rows | Extra                       |
+----+-------------+----------+-------+-------------------------------------+--
-------+---------+------+------+-------------------------+
|  1 | SIMPLE      | example1 | range | TXindex,comboTR,comboRT,comboRTD |
comboRTD | 5       | NULL | 4689 | USING WHERE; USING INDEX |
+----+-------------+----------+-------+-------------------------------------+--
-------+---------+------+------+-------------------------+
```

Лабораторная работа №6. Хранимые процедуры и функции

Цель занятия: освоить синтаксис хранимых процедур и триггеров.

Задание:

Создать хранимую процедуру с использованием оператора ветвления и цикла. Написать триггер, сохраняющий в таблицу время последнего изменения данных.

Ход работы.

Хранимая процедура— компилируемый объект базы данных, хранящийся на сервере и представляющий собой набор SQL-инструкций. Хранимые процедуры могут обладать:

- входными и выходными параметры;
- локальными переменными;
- циклами и ветвлениями, то есть в них могут использоваться инструкции управления потоком.
- в них могут производиться
  - стандартные операции с базами данных (как DDL, так и DML);
  - числовые вычисления;
  - операции над символьными данными;

  результаты которых могут присваиваться переменным и параметрам.

Создание

```
DELIMITER //
CREATE PROCEDURE `procedure2`(IN sTitle VARCHAR(255))
BEGIN
  INSERT INTO `threads` (`title`) VALUES (sTitle);
  UPDATE `variables` SET `value` = `value` + 1 WHERE `name` = 'threads';
END
// DELIMITER;
CREATE
   [DEFINER = { user | CURRENT_USER }]
   PROCEDURE sp_name ([proc_parameter[,...]])
```

```
    [characteristic ...] routine_body

CREATE
    [DEFINER = { user | CURRENT_USER }]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

proc_parameter:
    [IN | OUT | INOUT] param_name type
```

Параметр IN передает значение в процедуру. Процедура могла бы изменять значение, но модификация не видима вызывающему оператору, когда процедура завершается. Параметр OUT передает значение из процедуры обратно вызывающему оператору. Начальное значение внутри процедуры NULL, и значение видимо вызывающему оператору, когда процедура завершается. Параметр INOUT инициализирован вызывающим оператором, может изменяться процедурой, и любое изменение, сделанное процедурой, видимо вызывающему оператору, когда процедура завершается.

Вызов

```
CALL `procname`;
CALL `procname`(arg1,arg2,... argN);
SELECT `function_name` (arg1);
```

Удаление

```
DROP procedure `procname`;
DROP FUNCTION `functname`;
```

Разделители

```
delimiter //
delimiter ;;;
delimiter |
```

Переменные

Локальные переменные

```
DECLARE iVar INT DEFAULT 0;
SET iVar = 5;
SELECT * FROM `data` WHERE `id` = iVar;
```

где iVar - название переменной, INT - тип данных переменной, 0 - значение по умолчанию.

Присвоение переменной значения из запроса (если запрос возвращает более одной строки, то необходимо использовать limit)

```
DECLARE iVar INT DEFAULT 0;
SELECT COUNT(*) INTO iVar FROM `data`;
```

Пользовательские переменные

```
SET @iVar = 5;
SELECT @iVar;
```

Конструкции и управление потоком

IF THEN

```
IF search_condition THEN statement_list
    [ELSEIF search_condition
    THEN statement_list] ...
    [ELSE statement_list]
END IF
```

Пример:

```
DELIMITER //

CREATE FUNCTION SimpleCompare(n INT, m INT)
  RETURNS VARCHAR(20)

  BEGIN
    DECLARE s VARCHAR(20);

    IF n > m THEN SET s = '>';
    ELSEIF n = m THEN SET s = '=';
    ELSE SET s = '<';
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m);

    RETURN s;
  END

// DELIMITER ;


SELECT SimpleCompare('4','2');
-> //
+------------------------+
| SimpleCompare('4','2') |
+------------------------+
| 4 > 2                  |
+------------------------+
```

CASE

```
CASE WHEN search_condition THEN statement_list
        [WHEN search_condition THEN statement_list] ...
        [ELSE statement_list]
   END CASE
```

LOOP

```
[begin_label:]
   LOOP
    statement_list
```

```
   END LOOP
   [end_label]
```

REPEAT

```
[begin_label:]
  REPEAT statement_list
    UNTIL search_condition
  END REPEAT;
[end_label]
```

WHILE

```
[begin_label:]
  WHILE search_condition DO statement_list
  END WHILE
[end_label]
```

LEAVE и ITERATE

```
LEAVE label

ITERATE label
```

Инструкция LEAVE используется, чтобы из выйти любой помеченной конструкции управления потоком данных. Это может использоваться внутри BEGIN ... END или же конструкций цикла (LOOP, REPEAT, WHILE). А инструкция ITERATE может появляться только внутри инструкций LOOP, REPEAT и WHILE. ITERATE предназначена для повторного выполнения цикла.

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1;
    END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END
```

Пример

```
CREATE procedure `srednkvadr` ()
  begin
     declare sredn float DEFAULT 0;
     SELECT avg(val) INTO sredn FROM sko;
     UPDATE `sko` SET `sko`.`sko`= pow((`sko`.`val`-sredn),2);
  end;//
```

Триггеры

Триггер - особый вид хранимой процедуры, срабатывающий автоматически по наступлению события, связанного с модификацией данных в таблице, к которой данный триггер относится. Срабатывают по наступлению событий:

- insert;
- update;
- delete.

Триггеры могут выполняться до и после выполнения событий, для которых они установлены. Соответственно, на каждую таблицу могут быть установлены максимум 6 триггеров (по одному before и after для insert, delete, update).

Создание

```
delimiter |

CREATE TRIGGER testref AFTER INSERT ON test1
  FOR EACH ROW BEGIN
    INSERT INTO test2 (a2) VALUES (NEW.a1);
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
  END;
|

delimiter ;
```

Удаление

```
DROP TRIGGER [IF EXISTS] trigger_name
```

Лабораторная работа №7. Программные средства для работы с MySQL

Цель занятия: изучить средства, помогающие в работе с СУБД.

Задание:

Изучить программы MySQL Query Browser, MySQL Administrator, MySQL Workbench.

Ход работы.

MySQL Query Browser

MySQL Query Browser — инструмент для создания выполнения и оптимизации запросов.

Установка

1. Ubuntu Linux: sudo apt-get install mysql-gui-tools-common
2. Другие Linux и Windows: http://dev.mysql.com/downloads/gui-tools/5.0.html

Запуск

- Host name : localhost
- User name: root
- Password: ******

Интерфейс

1. Query Toolbar (панель запросов).
2. Advanced Toolbar (дополнительная панель).
3. Result Area (область результатов).
4. Object Browser (обозреватель объектов).

5. Information Browser (обозреватель информации).

Использование

Откройте вкладку «Results», перенесите мышкой из обозревателя объектов требуемые вам объекты (базы данных, таблицы, функции, синтаксические выражения и т.д.) в панель запросов. Таким образом вы сконструируете запрос.

При выборе таблиц и баз данных под панелью запросов станет доступны кнопки для изменения, добавления и удаления данных.

Для выполнения запроса нажмите кнопку «Выполнить» (Execute Query). Результат появится в области результатов.

MySQL Administrator

MySQL Administrator — инструмент для администрирования сервера MySQL (конфигурация, мониторинг, управление пользователями и подключениями, резервное копирование и другое).

Установка

1. Ubuntu Linux: sudo apt-get install mysql-gui-tools-common
2. Другие Linux и Windows: http://dev.mysql.com/downloads/gui-tools/5.0.html

Интерфейс

1. Sidebar (панель).
2. Working Area (рабочая область).

Вкладки

1. Server Information.
2. Service Control.
3. Startup Parameters.
4. User Administration.
5. Server Connections.
6. Health.
7. Server Logs.
8. Backup.
9. Replication Status.
10. Catalogs.

Открыв вкладки, вы увидите информацию о сервере по соответствующей тематике.

MySQL Workbench

MySQL Workbench — инструмент визуального проектирования баз данных. Позволяет проектировать, моделировать, создавать и поддерживать базы данных для MySQL.

Установка

1. Ubuntu Linux: ftp://ftp.mysql.com/pub/mysql/download/gui-tools/ubuntu/pool/wb51/m/mysql-workbench/
2. Другие Linux и Windows: http://dev.mysql.com/downloads/workbench/5.0.html

Создание ER-диаграмм

Построение новой ER-диаграммы начинается с создания нового проекта (File -> New Project).

Далее добавляем к проекту новую ER-диаграмму («Add Diagram»). Создаем в ней новые элементы: Таблицы (T), представления (V) и другие. При создании объектов описываем их свойства. Например, для таблицы указываются ее атрибуты и свойства атрибутов. Проводим между таблицами связи от поля одной таблицы до внешнего ключа второй таблицы. Тип связи (много ко многим, одно к одному и другие) определяется формой стрелки примитива.

При проектировании можно использовать уже готовую базу данных из файла (SQL-дампа). В этом случае ER-диаграмма будет построена автоматически.

Работа с базами данных

Выбрав в окне объектов базу данных, мы можем производить над ними изменения. Для этого нужно выбрать соответствующее действие в разделе «Physical Schema» («Add Table», «Add View» и другие). После выбора действия откроется окно с вкладками описания действия. Например, для создания таблицы будет предложено ввести имя, тип таблицы, атрибуты таблицы, ключи таблицы, индексы, триггеры, добавить данные в таблицу и другое.

Рекомендуемая литература

Основная литература:

1. Дунаев В.В. Базы данных. Язык SQL для студента / В. В. Дунаев. - СПб. : БХВ-Петербург, 2006. - 279[1] с. : ил. - Предм. указ.: с. 275-279. - ISBN 5-94157-823-7

2. Сибилёв В.Д. Базы данных: учебное пособие / В. Д. Сибилёв ; Федеральное агентство по образованию, Томский государственный университет систем управления и радиоэлектроники, Кафедра автоматизированных систем управления. - Томск : ТУСУР, 2007. – 278[1] с. : ил., табл. - Библиогр.: с. 273-274.

3. Давыдова Е.М. Базы данных: Учебное пособие / Е. М. Давыдова, Н.А. Новгородова ; Федеральное агентство по образованию, Томский государственный университет систем управления и радиоэлектрони Кафедра комплексной информационной безопасности электронно-вычислительных систем. - 2-е изд., перераб. и доп. - Томск : В-Спектр, 2007. - 127[1] с. : ил., табл. - Библиогр.: с. 114.

Дополнительная литература

1. Рудикова Л.В. Базы данных: Разработка приложений : Практическое руководство / Л. В. Рудикова. - СПб. : БХВ-Петербург, 2006. - 487[1] с. : ил., табл. - (Для студента). - Библиогр.: с. 481-482. - Предм. указ.: с. 483-487. - ISBN 5-94157-805-9

2. Крёнке Д.М. Теория и практика построения баз данных : Пер. с англ. / Д. М. Крёнке ; пер. А. Вахитов. - 9-е изд. - СПб. : Питер, 2005. - 858[6] с. : ил. - (Классика Computer Science). - Алф. указ.: с. 845-858. - ISBN 5-94723-583-8

3. Харрингтон Д. Разработка баз данных : Пер. с англ. / Д. Харрингтон. - М. : ДМК Пресс, 2005. - 269[1] с. : ил., табл. - (Специалист). - Предм. указ.: с. 267-269. - ISBN 5-94074-292-0 (в пер.)