***Лабораторная работа №2 (4 часа – 2 пары)***

**Тема:** Знакомство с MySQL. Создание и заполнение таблиц.

**Цель:** Получить практические навыки работы в MySQL создания и заполнения таблиц БД

**Задание:**

1. Изучить теоретическую часть.
2. Изучить ПрО в соответствии вариантом (по журналу) и построить схемы ПрО: IDEF0 с декомпозицией и DFD. Сопроводить схемы описательной частью.
3. Построить концептуальную, логическую и физическую схемы БД.
4. Используя SQL-запросы создать БД, таблицы и заполнить их данными (минимум 4 записи в главной и 8 – в зависимой).
5. Оформить отчет (по ГОСТ).

***Теоретическая часть***

**Создание баз данных средствами MySQL**

**Типы данных MySQL Строковые типы**

В MySQL определены следующие строковые типы (табл. 1).

Таблица 1. Тип столбца и требуемый объем памяти

|  |  |
| --- | --- |
| **Тип столбца** | **Необходимый объем памяти** |
| CHAR (M) | M байтов, 0≤ M ≥255 |
| VARCHAR(M) | L+1 байтов, где L≤ M и 0 ≤ M ≥ 65535 |
| BINARY (M) | М байтов, 0<M< 255 |
| VARBINARY(M) | L+1 байтов, где L≤ M и 0≤ M ≥ 255 |
| TINYBLOB, TINYTEXT | L+1 байтов, где L<28 |
| BLOB, TEXT | L+2 байтов, где L<216 |
| MEDIUMBLOB, MEDIUMTEXT | L+3 байтов, где L<224 |
| LONGBLOB, LONGTEXT | L+4 байтов, где L<232 |
| ENUM (‘value1’, ’value2’,…) | 1 или 2 байта в зависимости от числа значений (65535 значений максимум) |
| SET (‘value1’, ’value2’,…) | 1, 2, 3, 4 или 8 байтов в зависимости от числа значений (64 значения максимум) |

Типы **char** (строки фиксированной длины) и **varchar** (строки произвольной длины) — это типы, предназначенные для записи коротких фрагментов тек­ста. Длину каждого из них можно регулировать числом М. Записи в столбцы типа **char** будут дополняться пробелами до максимальной длины. При этом длина строки не зависит от размеров данных, в то время как в столбцах с ти­пом **varchar** размер поля зависит от размеров данных. MySQL удаляет про­белы в конце текстовых строк у **char** во время извлечения и у **varchar** во время сохранения.

Тип **varchar** позволяет экономить память, но при хранении данных прихо­дится указывать, сколько памяти занимает введенное значение. Это значение сервер использует для того, чтобы определить, с какой позиции начнется за­пись следующего значения. При работе с типом **char** не требуется указывать в каждом поле реальную длину строки, поэтому сервер работает с такими строками быстрее, зато используется несколько больший объем памяти.

Тип **blob** (Binary Large Object, большой двоичный объект) используется для хранения двоичных данных.

Данные типа **enum** могут принимать одно из нескольких заранее заданных значений: enum(value1,value2, ...). Например, предусмотрено, что в неко­тором столбце должен храниться ответ на вопрос, а допустимыми значения­ми ответа являются "да" и "нет". Тогда valuel будет "да", a value2 — "нет".

Тип **set** предусматривает, что в поле одновременно может содержаться несколько из заранее заданного списка значений.

**Форматы записи даты и времени**

Дату и время можно хранить в одном из следующих форматов (табл. 2).

Таблица 2. Типы значений даты и времени

|  |  |  |
| --- | --- | --- |
| **Тип столбца** | **Описание** | **Диапазон значений'** |
| DATETIME | YYYY-MM-DDHH:ММ:  :SS | '1000-01-01 00:00:00' — '9999-12-31 23:59:59' |
| DATE | YYYY-MM-DD | '1000-01-01' —'9999-12-31' |
| TIMESTAMP | YYYYMMDDHHMMS S | '1970-01-01 00:00:00' — '2037-12-31 23:59:59' |
| TIME | HH:MM:SS | '-838:59:59' — '838:59:59' |
| YEAR | YY ИЛИ YYYY | 1970-2069 или 1901-2155 |

Отсчет времени в MySQL осуществляется с начала эпохи UNIX — 1 января 1970 г.

**Хранение числовых значений**

Целые числа могут храниться со знаком (**signed**) и без (**unsigned**). Допусти­мые форматы целых чисел представлены в табл. 3.

Таблица 3. Целые числа

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Тип** | **Память в байтах** | **Минимальное значение** | | **Максимальное значение** | |
| **Signed** | **Unsigned** | **Signed** | **Unsigned** |
| TINYINT | 1 | -128 | 0 | 127 | 255 |
| SMALLINT | 2 | -32 768 | 0 | 32 767 | 65 535 |
| MEDIUMINT | 3 | -8 388 608 | 0 | 8 388 607 | 16 777 215 |
| INT | 4 | -2 147483 648 | 0 | 2147483 647 | 4 294 967 295 |
| BIGINT | 8 | -263 | 0 | 263-1 | 264-1 |

Числа с плавающей точкой могут храниться в столбцах следующих типов (табл. 4).

Таблица 4. Числа с плавающей точкой

|  |  |
| --- | --- |
| **Тип** | **Память в байтах** |
| FLOAT(р) | 4, если 0 < р < 24; 8, если 25 < р < 53 |
| FLOAT | 4 |
| DOUBLE *[размерность]* | 8 |
| DECIMAL*(М,D)*  NUMERIC*(М,D)* | Переменная |

Данные типа **float** или **double** округляются при записи до указанной в опре­делении столбца точности.

Тип **decimal** предназначен для хранения данных, которые не могут быть округлены, например, денежных величин. В MySQL версии 5.0.2 и более ранних данные этого типа хранятся в виде текста. Начиная с версии 5.0.3, данные типа **decimal** хранятся в двоичном виде, в котором десять двоичных цифр записываются в 4-х байтах.

**Работа с клиентской программой mysql**

Утилита **mysql** позволяет осуществить соединение с сервером MySQL и давать команды SQL для создания баз, записи в них данных и выборки данных из баз. В версии MySQL 5 реализован оконный интерфейс для рабо­ты с mysql. Его можно вызвать, выбрав пункт меню MySQL command line client в группе программ MySQL.

При запуске утилита mysql установит соединение с сервером и предложит ввести пароль суперпользователя **root**.

Команда на языке SQL обычно состоит из выражения, за которым следует точка с запятой. 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.

Строка **mysql>** называется приглашением командной строки.

Утилита mysql выводит результаты работы запроса, если таковые есть, 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.

Утилита mysql также сообщает количество возвращаемых строк и время вы­полнения запроса, что позволяет в некоторой степени составить представле­ние о производительности сервера.

Для mysql признаком завершения команды является точка с запятой, а не ко­нец строки. Команды собираются, но не исполняются до тех пор, пока про­грамма не обнаружит точку с запятой. Вот пример команды, требующей рас­печатать список баз данных сервера и занимающей несколько строк:

*mysql> SHOW*

*-> DATABASES;*

В табл. 5 приведены все возможные варианты вида метки командной стро­ки и соответствующие им состояния **mysql**

Таблица 5. Метка командной строки MySQL

|  |  |
| --- | --- |
| **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 |

**Создание базы данных**

На примере базы данных, предлагающей услуги такси рассмотрим процесс создания БД и таблиц средствами MySQL.

Создадим базу данных taxi таксопарка командой **create database**:

mysql>**CREATE DATABASE** *taxi*;

Получив команду, сервер выполняет ее и выдает со­общение (при успехе— Query OK, 1 row affected).

При создании базы данных она автоматически не выбирается, выбирать ее нужно отдельно командой **USE *name***  или **\u** *name*

mysql>**USE** *taxi*;

Таблицы создаются командой **create table**. При создании нужно указать не только имя таблицы, но и ее полное определение, состоящее из определений отдельных полей. Таблица **cars** (автомобили) должна содержать поля:

*model* — название модели автомобиля;

*madein* — года ее выпуска;

*reg\_number* — государственного регистрационного номера;

*color* — цвета

id - однозначно идентифицирую­щий запись об автомобиле — первичный ключ. Значения этого поля не должны повторяться и не могут быть пустыми, желательно, чтобы эти зна­чения были целыми числами, увеличивающимися на единицу в каждой но­вой записи.

*mysql>***CREATE TABLE** *cars*

*( model* CHAR(50) NOT NULL*,*

*madein* YEAR(4) NOT NULL*,*

*reg\_number* CHAR(12) NOT NULL*,*

*color* CHAR(15) NOT NULL*,*

*id* SMALLINT AUTO\_INCREMENT, PRIMARY KEY(id)

*);*

Атрибут **not null** означает, что все строки таблицы должны иметь значе­ние в этом столбце. Если not null не указано, поле может быть пустым (null).

**Primary key** после имени столбца определяет, что этот столбец является пер­вичным ключом для таблицы. Данные в этом столбце должны быть уникаль­ными.

**Auto\_increment** — атрибут, используемый для создания столбца с уникаль­ными значениями. Если при вставке строк в таблицу оставлять такое поле пустым, MySQL автоматически генерирует уникальное значение идентифи­катора. Это значение будет на единицу больше максимального значения, уже существующего в столбце. В каждой таблице может быть не больше одного такого поля.

MySQL использует следующий алгоритм для инициализации счетчика для столбца id, имеющего атрибут auto\_increment: после запуска сервера MySQL при первом запросе на добавление данных в таблицу cars сервер вы­считывает максимальное значение в столбце id. Полученное значение увели­чивается на единицу, заносится в новую запись и в счетчик. Если таблица была пуста, то счетчик устанавливается в единицу.

Таблица *drivers* содержит столбцы для хранения имени, отчества и фамилии водителя — *name*, *second\_name*, *family\_name*; даты его рождения *birth*, до­машнего адреса *address* и даты поступления на работу *startdate*.

mysql>**CREATE TABLE** *drivers* (

*name* CHAR(30) NOT NULL,

*second\_\_name* CHAR(30) ,

*family\_name* CHAR(30) NOT NULL,

*birth* DATE NOT NULL,

*address* CHAR(150) NOT NULL DEFAULT 'unkonwn',

*startdate* DATE NOT NULL DEFAULT '2002-01-01',

*id* SMALLINT AUTO\_INCREMENT,

PRIMARY KEY(id)

);

Кроме этого, при создании таблицы для некоторых полей могут применяться дополнительные ключевые слова, уточняющие диапазон возможных значе­ний. Ключевое слово **unsigned** может применяться после указания целочис­ленного типа и означает, что его значение может быть либо положительным, либо нулевым.

**default** — после этого ключевого слова указывается значение по умолчанию для данного столбца. Например:

**DEFAULT** 'no description'

Таблица *timetable* состоит из полей *use\_on* с датой, номером машины

*car\_number* и идентификатором водителя *driver\_number*.

mysql>**CREATE TABLE** *timetable* (

*use\_on* DATE DEFAULT NULL,

*car\_number* SMALLINT,

*driver\_number* SMALLINT,

*id* SMALLINT AUTO\_INCREMENT,

PRIMARY KEY(id),

INDEX (car\_number),

INDEX (driver\_number),

FOREIGN KEY (car\_number) REFERENCES cars (id),

FOREIGN KEY (driver\_number) REFERENCES drivers (id)

);

MySQL поддерживает ссылочную целостность базы с помощью ограничения внешнего ключа **foreign key**. Назначение этого ключа — проверять соответ­ствие значений в столбцах родительского и внешнего ключей. Эти столбцы должны иметь одинаковый тип данных, и только те значения, которые встре­чаются в родительском ключе, могут использоваться во внешнем ключе. Ро­дительский ключ должен содержать неповторяющиеся значения и не содер­жать значений **null**.

Последние два столбца таблицы *timetable* являются внешними ключами, и MySQL 5 требует, чтобы перед созданием внешних ключей на этих столб­цах были созданы индексы.

Индекс формируется из значений одного или нескольких столбцов таблицы и позволяет находить нужную строку по заданному значению. Для ускорения запросов индексы обычно создаются на тех столбцах таблицы, которые часто используются в запросах.

После создания таблиц можно просмотреть их список и структуру.

Посмотреть список таблиц в базе можно командой

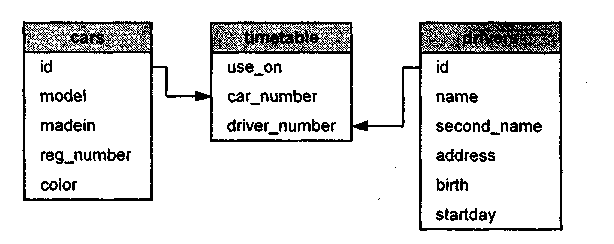
mysql>**SHOW TABLES**;

а структуру таблицы —

mysql>**DESCRIBE** *table\_name*;

Эта команда позволит посмотреть типы данных столбцов и дополнительные атрибуты, указанные при создании таблицы.

Схема базы данных *taxi* представлена на рис. 1.



**Запись данных в таблицы**

Для внесения данных в базу данных можно использовать оператор **insert**.

mysql> **INSERT INTO** *cars* VALUES ('Volga','2003','A786YC78','white','1');

Аналогично можно добавлять данные только в избранные поля, при этом надо указать имена полей явно:

mysql> **INSERT INTO** *cars* (*model*, *madein*, *reg\_\_number*, *color*) VALUES

-> ('Volga','2002','A788YC78','yellow');

mysql> **INSERT INTO** *cars* (*model*, *madein*, *reg\_number*, *color*) VALUES

-> ('Volkswagen','2003','A789YC78','red');

mysql> **INSERT INTO** *cars* (*model*, *madein*, *reg\_number*, *color*) VALUES

-> ('Renault','2005','A790YC78','white');

Добавим еще несколько значений в другие таблицы:

mysql> *INSERT INTO* *drivers* VALUES (

-> 'Stanislav', 'Petrovitch', 'Nikolaev',

-> '1970-01-22', 'Nevsky prospect 11, flat 54, '2004-02-30',1);

mysql> **INSERT INTO** *drivers* (*name*, *second\_name*, *family\_name,* *birth*,

-> *address*, *startdate*)

-> VALUES ('Fedor', 'Sergeevitch', 'Sentsov',

-> '1960-11-02', 'Moskovsky prospect 92, flat 112', '2003-12-30');

mysql> **INSERT INTO** *timetable* VALUES ('2006-08-09', 1,1), ('2006-07-21', 2,2), ('2006-07-20', 2,2);

Можно использовать встроенную в MySQL функцию **curdate()**, которая возвращает текущую дату в нужном формате:

mysql> **INSERT INTO** *timetable* VALUES (curdate(), 2,2);

Если вы попытаетесь оставить пустыми поля, отмеченные **not null** и не имеющие значения по умолчанию, то сервер MySQL выдаст сообщение об ошибке и не выполнит команду.

Кроме того, сервер MySQL 5 проверяет связи между таблицами при внесении записей. То есть вам не удастся внести в поле, являющееся внешним ключом, значение, отсутствующее в связанной таблице.

**Содержание отчета:**

* Тема;
* Цель;
* Задание (в соответствии с вариантом);
* Анализ ПрО и построение соответствующих схем (см. задание);
* SQL-запросы и результаты работы.

**Варианты (не должны повторяться в пределах группы)**

1. Учет выданных подарков несовершеннолетним детям сотрудников
2. Учет выполненных ремонтных работ
3. Продажа цветов
4. Поступление лекарственных средств
5. Списание оборудования
6. Поваренная книга
7. Регистрация входящей документации
8. Увольнение сотрудника
9. Приказ на отпуск
10. Регистрация выходящей документации
11. Выдача оборудования в прокат
12. Прием товаров в магазин
13. Учет оптовых продаж магазина
14. Учет подписки на периодические печатные издания
15. Учет сделок с недвижимостью
16. Учет результатов сдачи вступительных экзаменов
17. Парикмахерская
18. Мастерская
19. Туристическое агентство
20. Прокат автомобилей
21. Списание оборудования из проката
22. Поликлиника
23. Прием на работу
24. Учет нарушений правил ПДД
25. Учет договоров страхования
26. Станция технического обслуживания автомашин