Федеральное агентство по образованию

Государственное образовательное учреждение высшего профессионального образования

«**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**»

Утверждаю

Декан (директор)

С. А. Гайворонский

« » 2009 г.

**А.А. Пономарев**

**Создание структуры БД и манипулирование данными в СУБД MySQL**

Методические указания к выполнению лабораторных работ

по курсу «Базы данных» для студентов III курса специальностей 230105 «Программное обеспечение вычислительной техники и автоматизированных систем». 230201 «Информационные системы и технологии».

Издательство

Томского политехнического университета

2009

УДК 681.324.016

ББК 00000

П00

**Пономарев А.А.**

П00 Базы данных: методические указания по выполнению лабораторного практикума для студентов специальностей 230105 **«**Программное обеспечение вычислительной техники и автоматизированных систем**»**, 230201 **«**Информационные системы и технологии**».** В трех частях, часть первая. /Сост. А.А. Пономарев. – Томск: Изд-во ТПУ, 2009. – 80 с.

**УДК 681.324.016**

**ББК 00000**

Методические указания рассмотрены и рекомендованы   
к изданию методическим семинаром кафедры   
автоматики и компьютерных систем

« 29 » октября 2009 г.

Зав. кафедрой АиКС

доктор технических наук \_\_\_\_\_\_\_\_\_\_*Г.П. Цапко*

Председатель учебно-методической

комиссии \_\_\_\_\_\_\_\_\_\_*Е.А. Кочегурова*

*Рецензент*

Доктор технических наук, профессор

*Д.Г. Копаница*

© Пономарев А.А., 2009

©Томский политехнический университет, 2009

©Оформление. Издательство Томского   
политехнического университета, 2009

Оглавление

[Лабораторная работа №1. Интерфейс СУБД MySQL. Предоставление доступа и привилегий. Утилиты, входящие в состав СУБД 4](#_Toc246296692)

[Лабораторная работа №2. Моделирование баз данных средствами Erwin 16](#_Toc246296693)

[Лабораторная работа №3. Создание баз данных и таблиц в среде MYSQL. Информационное наполнение. 26](#_Toc246296694)

[Лабораторная работа №4 Создание запросов и модификация таблиц базы данных. 35](#_Toc246296695)

[Лабораторная работа №5. Работа с внешними базами данных. 46](#_Toc246296696)

[Ограничение доступа. 46](#_Toc246296697)

[Лабораторная работа №6. Представления, хранимые процедуры, функции, триггеры. 51](#_Toc246296698)

[Представления 51](#_Toc246296699)

[Хранимые процедуры и функции 52](#_Toc246296700)

[Триггеры 57](#_Toc246296701)

[Словарь данных 57](#_Toc246296702)

[Курсоры 60](#_Toc246296703)

# Лабораторная работа №1. Интерфейс СУБД MySQL. Предоставление доступа и привилегий. Утилиты, входящие в состав СУБД

**Цель работы:** Ознакомиться с приложениями, включенными в состав СУБД MySQL. Получить навыки управления учетными записями пользователей и определения привилегий. Ознакомиться с утилитами, входящими в состав СУБД MySQL, получить навыки работы с ними.

**Запуск MySQL**

Управление сервером обычно осуществляется из командной строки. Запуск в Windows 95/98/2000/XP осуществляется через сеанс DOS выполнением следующей команды:

|  |
| --- |
| D:\usr\local\Mysql\bin\mysqld --standalone |

Эта команда запустит демон mysql в фоновом режиме. В Windows 95/98 не предусмотрен запуск mysqld в виде службы. В Windows 2000 демон mysql запускается в виде службы.

Можно осуществить запуск winmysqladmin.exe, в этом случае все настройки перечисляются в файле my.ini

При запуске mysqld можно указывать следующие опции:

Таблица - Опции команды MySQLD

|  |  |
| --- | --- |
| -?, --help | Справка |
| -b, --basedir=[path] | Путь к каталогу в котором установлен mysql |
| -h, --datadir [homedir] | Путь к каталогу, в котором хранятся базы данных. |
| -l, --log=[filename] | Имя журнала транзакций |
| -L, --language=[language] | Язык по умолчанию(обычно English). |
| -P, --port=[port] | Порт для соединения. |
| --skip-grant-tables | Игнорировать таблицы привилегий. Это дает любому ПОЛНЫЙ доступ ко всем таблицам. Не следует предоставлять обычным пользователям разрешений на запуск mysqld. |
| --skip-name-resolve | Позволяет предоставлять доступ только тем хостам, чьи IP-адреса указаны в таблицах привилегий. Ипользуется для более высокого уровня защиты. |
| --skip-networking | Использовать подключения только через интерфейс localhost. |
| -V, --version | Вывести информацию о версии. |

Наличие в статусной строке иконки светофора с активным зеленым цветом указывает на то, что сервер запущен (см. рис 1).



Рисунок - Приложение winmysqladmin запущено

Теперь можно попытаться войти в сервер. В случае, если предполагается управление сервером через консоль, то необходимо использовать команду **mysql**. Изначально существует единственный пользователь, которому предоставляется право входа - **root**, которая не имеет пароля. Первое, что нужно сделать войти под именем **root** и зарегистрировать нового пользователя и установить для него пароль. Команда **mysql** может использовать следующие опции:

Таблица - Опции команды MySQL

|  |  |
| --- | --- |
| -?, --help | Справка |
| -h,--hostname=[hostname] | Имя сервера mysql. |
| -u, --user=[user] | Имя пользователя для доступа к mysql. |
| -p, --password=[password] | Пароль пользователя для доступа к mysql. |
| -P, --port=[port] | Порт для соединения с сервером. |
| -V, --version | Информация о версии |

**Примечание.** Команды mysqld и mysql имеют еще некоторые опции, но в данный момент они особого интереса не представляют.

Запуск из сеанса ДОС осуществляется как показано на Рисунок 2 (в указанном случае осуществляется подключение к БД mysql).

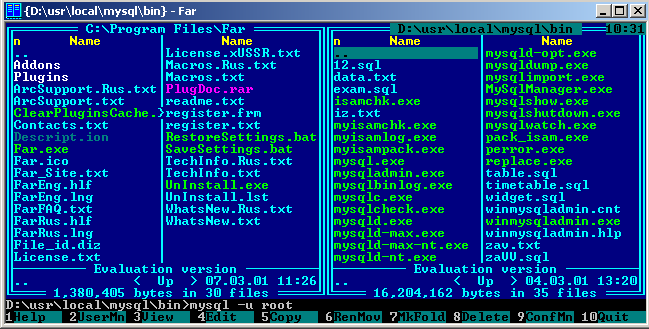


Рисунок - Запуск консоли MYSQL

Для выполнения в строке наберите команду: **mysql –u root**

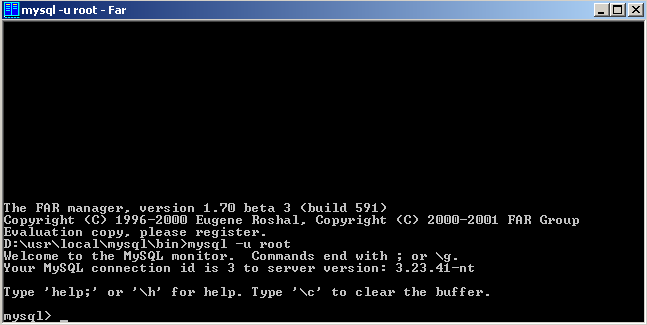


Рисунок - Успешный запуск консоли

Если вы это получили, значит вы успешно вошли в консоль mysql, которая используется для администрирования сервера.

Для составления отчета вам понадобятся приведение команд, которые вы будете посылать на сервер. В MySQL имеется возможность ведение протокола выполняемых команд, чтобы запустить ведение протокола необходимо выполнить команду

**\T filename**

!!! обязательно в верхнем регистре. Filename – имя файла, в который будут записываться команды (создается автоматически при выполнении команды, и действует во время жизни сеанса, т.е. в случае отключения от сервера лог прерывается и для возобновления необходимо повторить команду с выводом в новый файл, так как команда затирает имеющиеся в файле данные).

Просмотр списка БД, доступных на сервере осуществляется командой ***SHOW DATABASES.***

Для выполнения в строке наберите команду: **show databases.**

Командой: **USE MYSQL;** – выбираем текущую БД где MYSQL имя БД.

**Система привилегий и безопасность в MySQL**

* User
* Db
* Host
* Пользовательские привилегии

**База данных mysql и таблицы привилегий.**

Итак, вы успешно вошли в базу данных mysql, которая используется для администрирования сервера. Что же здесь находится? А находятся здесь 5 таблиц, которые ничем не отличаются от других таблиц баз данных, за исключением того, что эти таблицы используются для предоставления доступа к базам данных и таблицам в них пользователям. Рассмотрим каждую из них.

Введите следующую команду, **show tables*,***  которая покажет таблицы в базе данных mysql.

Кратко рассмотрим функции каждой из таблиц:

**Таблица User**

Определяет, разрешено ли пользователю, пытающемуся подключиться к серверу делать это. Содержит имя пользователя, пароль а также привилегии. Если ввести команду show columns from user; то получим следующее:

Таблица - Структура таблицы User

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| Host | char(60) |  | PRI |  |  |
| User | char(16) |  | PRI |  |  |
| Password | char(41) |  |  |  |  |
| Select\_priv | enum('N','Y') |  |  | N |  |
| Insert\_priv | enum('N','Y') |  |  | N |  |
| Update\_priv | enum('N','Y') |  |  | N |  |
| Delete\_priv | enum('N','Y') |  |  | N |  |
| Create\_priv | enum('N','Y') |  |  | N |  |
| Drop\_priv | enum('N','Y') |  |  | N |  |
| Reload\_priv | enum('N','Y') |  |  | N |  |
| Shutdown\_priv | enum('N','Y') |  |  | N |  |
| Process\_priv | enum('N','Y') |  |  | N |  |
| File\_priv | enum('N','Y') |  |  | N |  |
| Grant\_priv[[1]](#footnote-1) | enum('N','Y') |  |  | N |  |
| References\_priv | enum('N','Y') |  |  | N |  |
| Index\_priv | enum('N','Y') |  |  | N |  |
| Alter\_priv | enum('N','Y') |  |  | N |  |
| Show\_db\_priv | enum('N','Y') |  |  | N |  |
| Super\_priv | enum('N','Y') |  |  | N |  |
| Create\_tmp\_table\_priv | enum('N','Y') |  |  | N |  |
| Lock\_tables\_priv | enum('N','Y') |  |  | N |  |
| Execute\_priv | enum('N','Y') |  |  | N |  |
| Repl\_slave\_priv | enum('N','Y') |  |  | N |  |
| Repl\_client\_priv | enum('N','Y') |  |  | N |  |
| Create\_view\_priv | enum('N','Y') |  |  | N |  |
| Show\_view\_priv | enum('N','Y') |  |  | N |  |
| Create\_routine\_priv | enum('N','Y') |  |  | N |  |
| Alter\_routine\_priv | enum('N','Y') |  |  | N |  |
| Create\_user\_priv | enum('N','Y') |  |  | N |  |
| Event\_priv | enum('N','Y') |  |  | N |  |
| Trigger\_priv | enum('N','Y') |  |  | N |  |
| ssl\_type | enum('','ANY','X509','SPECIFIED') |  |  |  |  |
| ssl\_cipher | blob |  |  | NULL |  |
| x509\_issuer | blob |  |  | NULL |  |
| x509\_subject | blob |  |  | NULL |  |
| max\_questions | int(11) unsigned |  |  | 0 |  |
| max\_updates | int(11) unsigned |  |  | 0 |  |
| max\_connections | int(11) unsigned |  |  | 0 |  |
| max\_user\_connections | int(11) unsigned |  |  | 0 |  |

Изначально эта таблица содержит пользователя root без пароля. По умолчанию root может входить с любого хоста, имеет все привилегии и доступ ко всем базам данных.Также в таблице содержится запись для пользователя '%'.

В БД MYSQL содержатся таблицы, называемых таблицами привилегий. Система привилегий будет подробно рассмотрена в следующих работах, а пока вы можете выполнить команды на добавления своего пользователя:

Для добавления нового пользователя **your\_name**, можно выполнить следующие операторы языка (Insert):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***Insert into user (host, user, password, ssl\_cipher[[2]](#footnote-2), x509\_issuer, x509\_subject*** ***) values ('localhost’, 'your\_name', password('your\_pass'), ‘’, ‘’, ‘’);***  Выполнением команды  ***Select host, user, password from user;***  Мы выводим перечисленные поля в виде таблицы   |  |  |  | | --- | --- | --- | | Host | User | Password | | % | root | 456g879k34df9 | |

Если необходимо выделить все столбцы таблицы, то необходимо набрать \* в качестве аргумента команды ***select***.

Чтобы изменения вступили в силу нужно перегрузить сервер, предварительно закончив текущий сеанс работы командой ***quit***.

|  |
| --- |
| ***mysqladmin -u root reload*** (эта команда перегружает сервер) |

После установки пароля для пользователя нужно перезагрузить сервер командой mysqladmin reload, чтобы изменения вступили в силу. После этого можно попробовать войти снова:

|  |
| --- |
| Mysql/bin/***mysql -u your\_name -p mysql***  Enter password:\*\*\*\*\*\*\* |

Если же после этой операции вы не получите приглашение ко входу, то необходимо будет повторить вход в сервер под учетной записью **ROOT** и назначить необходимые права**.**  Т.о., недостаточно добавить сведения о пользователе в системную БД, дополнительно необходимо назначить права пользователю, после чего можно начинать настраивать таблицы привилегий, вводить новых пользователей, создавать базы данных и таблицы, то есть делать все то, что называется администрированием. Назначить права можно указанием инструкцией ***INSERT*** для заполнения соответствующие привилегии (перечень привилегий см. Таблица 3)

|  |
| --- |
| Mysql/bin/***mysql -u root***  И выполнить следующий запрос к БД:  Mysql>***USE MYSQL;***  Mysql>***GRANT ALL PRIVILEGES ON \*.\* TO 'your\_name'@'localhost*[[3]](#footnote-3)*' IDENTIFIED BY 'your\_pass' WITH GRANT OPTION;***  Mysql>***FLUSH PRIVILEGES;*** |

Если пароль был случайно забыт, чтобы его задать по новой, придется стереть файлы mysql.frm mysql.MYI и mysql.MYD из папки с базами данных, затем запустить скрипт mysql\_install\_db и повторить все по новой. Можно воспользоваться ключом MYSQL и ввести **--skip-grant-tables,** при этом все пароли будут имеет пустое поле.

Команда имеет вид ***mysqld --skip-grant-tables.***

**Пояснения:**

1.Команда insert вставляет данные в таблицу, не забывайте завершать команды ';'.

2.При вводе пароля используйте функцию password(), иначе пароль работать не будет!

3.Все пароли шифруются mysql, поэтому в поле Password вы видите абракадабры. Это делается в целях безопасности.

4.Не есть хорошей практикой назначать привилегии пользователям в таблице user, так как в этом случае они являются глобальными и распространяются на все базы данных. Предоставляйте привилегии каждому пользователю к конкретной базе данных в таблице db, которая будет рассмотрена далее.

5.При задании имени хоста для входа через сеть рекомендуется явно указывать полное имя хоста, а не '%'. В приведенном выше примере пользователю mary разрешается вход на сервер со всех машин домена tomsk.ru. Можно также указывать IP-адреса машин и маски подсетей для большей безопасности.

**Таблица Db**

Определяет к каким базам данных каким пользователям и с каких хостов разрешен доступ. В этой таблице можно предоставлять каждому пользователю доступ к базам данных и назначать привилегии. Eсли выполнить команду ***show columns from db;*** получим следующее:

Таблица - Структура таблицы Db

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| Host | char(60) |  | PRI |  |  |
| Db | char(32) |  | PRI |  |  |
| User | 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 |  |

* По умолчанию, все привилегии установлены в 'N'. Например, предоставим юзеру mary доступ к базе данных mysql и дадим ему привилегии **select**, **insert** и **update** (описание основных команд mysql будет дано в следующих лабораторных работах, сейчас ваша цель увидеть, как работают таблицы привилегий).
* Для справки:

|  |
| --- |
| ***Insert into db (host, user, db, select\_priv, insert\_priv, update\_priv)***  ***Values (''localhost', 'your\_name', mysql, 'Y', 'Y', 'Y');*** |

* Привилегии, устанавливаемые в таблице db, распространяются только на базу данных library. Если же установить эти привилегии в таблице user, то они будут распространяться и на другие базы данных, даже если доступ к ним и не установлен явно.

**Таблица Host**

Таблица host используется для расширения диапазона доступа в таблице db. К примеру, если доступ к какой-либо базе данных должен быть предоставлен более чем одному хосту, тогда следует оставить пустой колонку host в таблице db, и внести в таблицу host необходимые имена хостов. Выполним команду

***show columns from host;***

Таблица - Структура таблиц Host

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field** | **Type** | **Null** | **Key** | **Default** | **Extra** |
| Host | 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 |  |

Как видно из таблицы, здесь также можно задавать привилегии для доступа к базе данных.Они обычно редко используются без необходимости. Все привилегии доступа нужно задавать в таблице db для каждого пользователя, а в таблице host только перечислить имена хостов.Сервер читает все таблицы, проверяет имя пользователя, пароль, имя хоста, имя базы данных, привилегии.Если в таблице db привилегии select, insert установлены в 'Y', а в таблице host в 'N', то в итоге юзер все равно получит 'Y'.Чтобы не вносить путаницы, лучше назначать привилегии в таблице db.

Эти 3 таблицы являются основными.В новых версиях MySQL, начиная с 3.22 добавлены еще 2 таблицы- tables\_priv и columns\_priv, которые позволяют задать права доступа к определенной таблице в базе данных и даже к определенной колонке. Они работают подобно таблице db, только ссылаются на таблицы и колонки. Также, начиная с версии 3.22 можно использовать команду GRANT для предоставления доступа к базам данных, таблицам и колонкам таблиц, что избавляет от необходимости вручную модифицировать таблицы db, tables\_priv и columns\_priv. Команда GRANT будет подробно рассмотрена в следующих разделах.

**Привилегии, предоставляемые MySQL**

Таблица - Привилегии пользователя[[4]](#footnote-4)

|  |  |  |
| --- | --- | --- |
| **Привилегия** | **Колонка** | **Где используется** |
| **select** | Select\_priv | таблицы |
| **insert** | Insert\_priv | таблицы |
| **Update** | Update\_priv | таблицы |
| **delete** | Delete\_priv | таблицы |
| **index** | Index\_priv | таблицы |
| **alter** | Alter\_priv | таблицы |
| **create** | Create\_priv | БД, таблицы, индексы |
| **drop** | Drop\_priv | БД или таблицы |
| **grant** | Grant\_priv | БД или таблицы |
| **References** | References\_priv | БД или таблицы |
| **reload** | Reload\_priv | администрирование сервера |
| **Shutdown** | Shutdown\_priv | администрирование сервера |
| **Process** | Process\_priv | администрирование сервера |
| **file** | File\_priv | доступ к файлам на сервере |

**Основные утилиты MySQL.**

В состав дистрибутива MySQL входят следующие утилиты:

* mysqld
* mysql
* mysqladmin
* mysqlaccess
* mysqlshow
* mysqldump
* isamchk

Утилиты **mysqld** и **mysql** были подробно рассмотрены ранее, поэтому возвращаться к ним не будем. Кратко рассмотрим остальные.

**Mysqladmin**

Утилита для администрирования сервера. Может использоваться администратором, а также некоторыми пользователями, которым предоставлены определенные привилегии, например – **Reload\_priv, Shutdown\_priv, Process\_priv** и **File\_priv**. Данная команда может использоваться для создания баз данных, изменения пароля пользователя(администратор может изменить пароль любому пользователю, а рядовой пользователь – только свой собственный), перезагрузки и остановки сервера, просмотра списка процессов, запущенных на сервере. Mysqladmin поддерживает следующие команды:

Таблица - Опции команды MySQLadmin

|  |  |
| --- | --- |
| Create [database\_name] | Создает базу данных |
| Drop [database\_name] | Удаляет базу данных и все таблицы в ней |
| Reload | Перезагружает сервер |
| Shutdown | Останавливает работу сервера MySQL |
| Processlist | Выводит список процессов на сервере |
| Status | Выводит сообщение о статусе сервера |

Пример использования mysqladmin для изменения пароля:

***mysqladmin -u your\_name password your\_pass***

Следует заметить, что в случае использования mysqladmin для установки пароля, не требуется использование функции password().Mysqladmin сам заботится о шифровании пароля.

**Mysqlaccess**

Используется для проверки привилегий пользователя для доступа к конкретной базе данных. Общий синтаксис:

***mysqlaccess [host] [user] [db] опции***

Полезная утилита для проверки прав доступа пользователя, если он получает сообщение Access denied, при попытке соединиться с базой данных.

**Опции:**

Таблица - Опции команды MySQLAccess

|  |  |
| --- | --- |
| -?, --help | Справка |
| -u, --user=[username] | Имя пользователя |
| -p, --password=[password] | Пароль пользователя |
| -h, --host=[hostname] | Имя хоста для проверки прав доступа |
| -d, --db=[dbname] | Имя базы данных для проверки прав доступа |
| -U, --superuser=[susername] | Имя суперпользователя(root) |
| -P, --spassword=[spassword] | Пароль администратора |
| -b, --brief | Выводит краткие сведения о таблице |

**Mysqlshow**

Используется, чтобы показать, с какими базами данных работает сервер, какие таблицы содержит каждая БД и какие колонки есть в каждой таблице.Синтаксис:

***mysqlshow [опции] [database [table [field]]]***

Mysqlshow может использовать следующие параметры:

Таблица - Параметры команды Mysqlshow

|  |  |
| --- | --- |
| -?, --help | Справка |
| -h, --host=[hostname] | Имя сервера |
| -k, --key | Показать ключи для таблицы |
| -p, --password=[password] | Пароль пользователя |
| -u, --user=[username] | Имя пользователя |
| -p, --port=[port] | Порт для связи |
| -V, --version | Вывести информацию о версии |

Если ввести mysqlshow без аргументов, будут показаны все базы данных, если указать имя БД, будут показаны все таблицы в ней.

Команды

***mysqlshow***

***mysqlshow mysql***

**Mysqldump**

Программа mysqldump используется для создания дампа содержания базы данных MySQL. Она пишет инструкции SQL в стандартный вывод. Эти инструкции SQL могут быть переназначены в файл. Можно резервировать базу данных MySQL, используя mysqldump, но при этом Вы должны убедиться, что в этот момент с базой данных не выполняется никаких других действий. А то mysqldump Вам такого нарезервирует...

Программа mysqldump поддерживает следующие параметры (Вы можете использовать короткую или подробную версию):

Таблица - Опции команды MySQLdump

|  |  |
| --- | --- |
| -#, --debug=[options] | **Вывести в протокол отладочную информацию. В общем виде 'd:t:o, filename`.** |
| -?, --help | Справка. |
| -c, --compleat-insert | Генерируйте полные инструкции insert (не исключая значений, которые соответствуют значениям столбца по умолчанию). |
| -h, --host=[hostname] | Соединиться с сервером hostname. |
| -d, --no-data | Экспорт только схемы информации (исключая данные). |
| -t, --no-create-info | Экспорт только данных, исключая информацию для создания таблицы. Противоположность -d. |
| -p, --password=[password] | Пароль пользователя, для соединения с сервером MySQL. Обратите внимание, что не должно быть пробела между -p и паролем. |
| -q, --quick | Не буферизовать результаты запроса, дамп выдать непосредственно к STDOUT. |
| -u, --user=[username] | Имя пользователя. Если не задано, используется текущий логин. |
| -v, --verbose | Вывести подробную информацию относительно различных стадий выполнения mysqldump. |
| -P, --port=[port] | Порт для связи. |
| -V, --version | Информация о версии. |

Вы можете направить вывод mysqldump в клиентскую программу MySQL, чтобы копировать базу данных. ПРИМЕЧАНИЕ: Вы должны убедиться, что база данных не изменяется в это время, иначе Вы получите противоречивую копию!

Для справки:

**mysqldump -u root -p mysql user>mysql-1.sql**

**mysqldump -u root mysql>mysql-2.sql**

**Примечание** флаг –p используется в случае, если пользователь наделен паролем.

После выполнения этой команды у нас появился файл mysql-1.sql и mysql-2.sql. Загрузим их в текстовый редактор, чтобы поподробнее изучить, и, возможно, немного поправить.

**Задание**

Запустите сервер MySQL. Зарегистрируйте своего пользователя в консольном приложении, задайте ему права.

С помощью утилиты Mysqlshow выполните команду на просмотр структуры и состав таблиц базы Mysql. Приведите в отчете её схему. С помощью утилиты Mysqldump получите полный дамп базы Mysql (данные и таблицы), а также отдельные дампы таблиц и данных.

**Контрольные вопросы:**

1. Каким способом возможен запуск серверной части СУБД.

2. Что такое привилегия. Каково её предназначение.

3. Какие основные утилиты входят в состав СУБД, какие функции они выполняют.

# Лабораторная работа №2. Моделирование баз данных средствами Erwin

**Цель работы** – приобретение студентами практических навыков создания логических и физических моделей данных с помощью CASE – средств разработки информационных систем.

**Основные сведения**

Система ERwin поддерживает прямое и обратное моделирование баз данных. При прямом моделировании схема базы данных описывается в прямом виде с использованием диаграммы сущность-связь. Сущности на диаграмме представляются прямоугольниками. Каждый прямоугольник может иметь различные визуальные атрибуты. Каждой сущности должно быть присвоено уникальное имя. Имена сущностей необходимо задавать в единственном числе. Это определяется тем, что система всегда оперирует отдельными экземплярами сущности. При этом отдельные экземпляры сущности рассматриваются как объекты, а сущности – как класс объектов. Если сущности были описаны при моделировании в BPwin, то их можно просто импортировать в ERwin. Пример диаграммы с созданными сущностями приведен на рисунке.

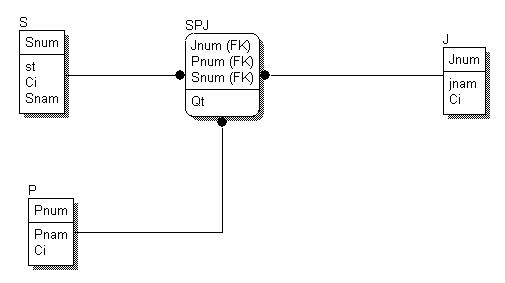


Рисунок - Пример диаграммы с созданными сущностями

**Построение моделей в ERwin**

Возможны две точки зрения на информационную модель и, соответственно, два уровня модели. Первый - логический уровень (точка зрения пользователя) означает прямое отображение фактов из реальной жизни. Например, люди, столы, отделы, собаки и компьютеры являются реальными объектами. Они именуются на естественном языке, с любыми разделителями слов (пробелы, запятые и т.д.). На физическом уровне модели рассматривается использование конкретной СУБД, определяются типы данных (например, целое или вещественное число), индексы для таблиц.

ERwin предоставляет возможности создавать и управлять этими двумя различными уровнями представления одной диаграммы (модели), равно как и иметь много вариантов отображения на каждом уровне. Термин "логический уровень" в ERwin соответствует концептуальной модели.

**Этапы построения информационной модели.**

* определение сущностей;
* определение зависимостей между сущностями;
* задание первичных и альтернативных ключей;
* определение атрибутов сущностей;
* приведение модели к требуемому уровню нормальной формы;
* переход к физическому описанию модели: назначение соответствий имя сущности - имя таблицы, атрибут сущности - атрибут таблицы;
* задание триггеров, процедур и ограничений;
* генерация базы данных.

Erwin создает визуальное представление (модель данных) для решаемой задачи. Это представление может использоваться для детального анализа, уточнения и распространения документации, необходимой в цикле разработки. Однако ERwin далеко не только инструмент для рисования. ERwin автоматически создает базу данных (таблицы, индексы, хранимые процедуры, триггеры для обеспечения ссылочной целостности и другие объекты, необходимые для управления данными).

**Создание сущности.**

Для внесения сущности в модель необходимо щелкнуть по кнопке сущности на панели инструментов (Erwin Toolbox) , затем - по тому месту на диаграмме, где необходимо расположить новую сущность. Щелкнув правой кнопкой мыши по сущности и выбрав из всплывающего меню пункт Entity Editor, можно вызвать диалог Entity Editor, в котором определяются имя, описание и комментарии сущности.



Каждая сущность должна быть полностью определена с помощью текстового описания в закладке Definition. Эти определения полезны как на логическом уровне, поскольку позволяют понять, что это за объект, так и на физическом уровне, поскольку их можно экспортировать как часть схемы и использовать в реальной БД (**CREATE COMMENT on entity\_name**). Закладки Note, Note2, Note3, UDP (User Defined Properties - Свойства, определенные пользователем) служат для внесения дополнительных комментариев и определений к сущности.

В закладке Icon каждой сущности можно поставить в соответствие изображение, которое будет отображаться в режиме просмотра модели на уровне иконок и изображение, которое будет отображаться на всех других уровнях.

Закладка UDP диалога Entity Editor служит для определения свойств, определяемых пользователем (User - Defined Properties). При нажатии на кнопку этой закладки вызывается диалог User - Defined Property Editor (также вызывается из меню Edit/UDPs). В нем необходимо указать вид объекта, для которого заводится UDP (диаграмма в целом, сущность, атрибут и т.д.) и тип данных. Для внесения нового свойства следует щелкнуть в таблице по кнопке и внести имя, тип данных, значение по умолчанию и определение.



**Создание атрибутов.**

Следующий этап создания модели состоит в задании атрибутов для каждой сущности. При задании типа атрибута имеется возможность использовать домены. Домен – это абстрактный пользовательский тип, который присваивается любому физическому типу данных. При этом каждый домен может иметь свои значения по умолчанию и правила проверки вводимых данных. ERwin предоставляет возможность документировать все действия по созданию собственных типов данных. С использованием концепции домена обеспечивается переносимость базы данных на различные аппаратные платформы.

|  |  |
| --- | --- |
| Рисунок - Создание нового домена | Рисунок - Указание свойств нового домена |
|  |  |

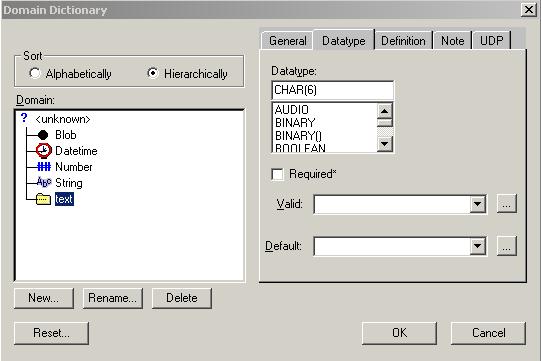


Рисунок - Значение по умолчанию для нового домена

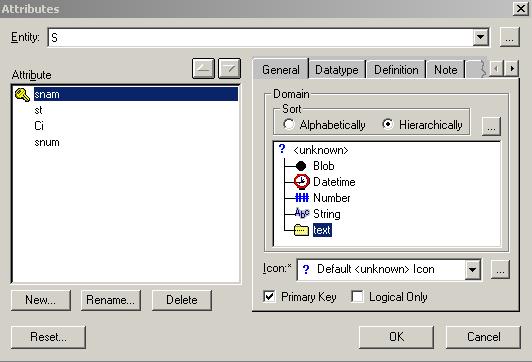


Рисунок - Использование домена для указания типа данных атрибуту.[[5]](#footnote-5)

Для описания атрибутов следует, щелкнув правой кнопкой по сущности, выбрать в появившемся меню пункт Attribute Editor. Появится диалог Attribute Editor.

Если щелкнуть по кнопке New, то в появившемся диалоге New Attribute можно указать имя атрибута, имя соответствующей ему в физической модели колонки и домен. Домен атрибута будет использоваться при определении типа колонки на уровне физической модели.

Для атрибутов первичного ключа в закладке General диалога Attribute Editor необходимо сделать пометку в окне выбора Primary Key.  
Закладки Definition, Note и UDP несут те же функции, что и при определении сущности, но на уровне атрибутов.

Для большей наглядности диаграммы каждый атрибут можно связать с иконкой. Это можно сделать при помощи списка выбора Icon в закладке General.

Очень важно дать атрибуту правильное имя. Атрибуты должны именоваться в единственном числе и иметь четкое смысловое значение.

Согласно синтаксису IDEF1X, имя атрибута должно быть уникальным в рамках модели (а не только в рамках сущности!). По умолчанию при попытке внесения уже существующего имени атрибута ERwin переименовывает его. Например, если атрибут Комментарий уже существует в модели, другой атрибут (в другой сущности) будет назван Комментарий/2, затем Комментарий/3 и т.д.  
При переносе атрибутов внутри и между сущностями можно воспользоваться техникой drag&drop, выбрав кнопку в палитре инструментов.



Для создания новой связи следует выбрать идентифицирующую или неидентифицирующую связь в палитре инструментов (ERwin Toolbox), щелкнуть сначала по родительской, а затем по дочерней сущности.  
В палитре инструментов кнопка соответствует идентифицирующей связи, кнопка связи многие-ко-многим и кнопка соответствует неидентифицирующей связи. Для редактирования свойств связи следует щелкнуть правой кнопкой мыши по связи и выбрать на контекстном меню пункт Relationship Editor.



В закладке General появившегося диалога можно задать мощность, имя и тип связи.

Связи на диаграмме представляются линиями, идущими от одной сущности (таблицы) к другой. Каждой связи присваивается уникальное имя. Связанные таблицы разделяют на родительские и дочерние. Родительские таблицы отображаются прямоугольниками с прямыми углами, дочерние – со скругленными.

**Мощность связи (Cardinality)** - служит для обозначения отношения числа экземпляров родительской сущности к числу экземпляров дочерней.  
Различают четыре типа мощности:

* общий случай, когда одному экземпляру родительской сущности соответствуют 0, 1 или много экземпляров дочерней сущности, не помечается каким-либо символом;
* символом P помечается случай, когда одному экземпляру родительской сущности соответствуют 1 или много экземпляров дочерней сущности (исключено нулевое значение);
* символом Z помечается случай, когда одному экземпляру родительской сущности соответствуют 0 или 1 экземпляр дочерней сущности (исключены множественные значения);
* цифрой помечается случай, когда одному экземпляру родительской сущности соответствует заранее заданное число экземпляров дочерней сущности.

По умолчанию символ, обозначающий мощность связи, не показывается на диаграмме. Для отображения имени следует в контекстном меню, которое появляется, если щелкнуть правой кнопкой мыши по любому месту диаграммы, не занятому объектами модели, выбрать пункт Display Options/Relationship и затем включить опцию Cardinality.

**Тип связи (идентифицирующая/неидентифицирующая).**

В IDEF1X различают зависимые и независимые сущности. Тип сущности определяется ее связью с другими сущностями. Идентифицирующая связь устанавливается между независимой (родительский конец связи) и зависимой (дочерний конец связи) сущностями. Когда рисуется идентифицирующая связь, ERwin автоматически преобразует дочернюю связь в зависимую. Зависимая сущность изображается прямоугольником со скругленными углами.

Экземпляр зависимой сущности определяется только через отношение к родительской сущности. При установлении идентифицирующей связи атрибуты первичного ключа родительской сущности автоматически переносятся в состав первичного ключа дочерней сущности. Эта операция дополнения атрибутов дочерней сущности при создании связи называется миграцией атрибутов. В дочерней сущности новые атрибуты помечаются как внешние ключи - (FK).

При установлении неидентифицирующей связи дочерняя сущность остается независимой, а атрибуты первичного ключа родительской сущности мигрируют в состав неключевых компонентов дочерней. Неидентифицирующая связь служит для связи независимых сущностей.

Идентифицирующая связь показывается на диаграмме сплошной линией с жирной точкой на дочернем конце связи, неидентифицирующая - пунктирной.

Для неидентифицирующей связи можно указать обязательность (Nulls в закладке General диалога Relationship Editor). В случае обязательной связи (No Nulls) при генерации схемы БД атрибут внешнего ключа получит признак NOT NULL, несмотря на то, что внешний ключ не войдет в состав первичного ключа дочерней сущности. В случае необязательной связи (Nulls Allowed) внешний ключ может принимать значение NULL. Необязательная неидентифицирующая связь помечается прозрачным ромбом со стороны родительской сущности

**Имя связи (Verb Phrase)** - фраза, характеризующая отношение между родительской и дочерней сущностями. Для связи один-ко-многим идентифицирующей или неидентифицирующей достаточно указать имя, характеризующей отношение от родительской к дочерней сущности (Parent-to-Child). Для связи многие-ко-многим следует указывать имена как Parent-to-Child, так и Child-to-Parent. Для отображения имени следует в контекстном меню, которое появляется, если щелкнуть правой кнопкой мыши по любому месту диаграммы, не занятому объектами модели, выбрать пункт Display Options/Relationship и затем включить опцию Verb Phrase.

**Имя роли или функциональное имя (Rolename)** - это синоним атрибута внешнего ключа, который показывает, какую роль играет атрибут в дочерней сущности. Задать имя роли можно в закладке Rolename/RI Actions диалога Relationship Editor.

**Создание ключей.**

Каждый экземпляр сущности должен быть уникален и отличаться от других атрибутов.

**Первичный ключ (primary key)** - это атрибут или группа атрибутов, однозначно идентифицирующие экземпляр сущности. Атрибуты первичного ключа на диаграмме не требуют специального обозначения - это те атрибуты, которые находятся в списке атрибутов выше горизонтальной линии. При внесении нового атрибута в диалоге Attribute Editor для того, чтобы сделать его атрибутом первичного ключа, нужно включить флажок Primary Key в нижней части закладки General. На диаграмме ключевой атрибут можно внести в состав первичного ключа, воспользовавшись режимом переноса атрибутов (кнопка в палитре инструментов).



В одной сущности может оказаться несколько атрибутов или наборов атрибутов, претендующих на роль первичного ключа. Такие претенденты называются **потенциальными ключами (candidate key).**

Ключи могут быть сложными, т.е. содержащими несколько атрибутов. Сложные первичные ключи не требуют специального обозначения - это список атрибутов выше горизонтальной линии. При выборе первичного ключа предпочтение должно отдаваться более простым ключам, т.е. ключам, содержащим меньшее количество атрибутов.

Многие сущности имеют только один потенциальный ключ. Такой ключ становится первичным. Некоторые сущности могут иметь более одного возможного ключа. Тогда один из них становится первичным, а остальные - альтернативными ключами.

**Альтернативный ключ (Alternative Key)** - это потенциальный ключ, не ставший первичным.

Каждому ключу соответствует индекс, имя которого также присваивается автоматически. Имена ключа и индекса при желании можно изменить вручную.

На диаграмме атрибуты альтернативных ключей обозначаются как (Akn.m.), где n - порядковый номер ключа, m - порядковый номер атрибута в ключе. Когда альтернативный ключ содержит несколько атрибутов, (Akn.m.) ставится после каждого.

**Внешние ключи (Foreign Key)** создаются автоматически, когда связь соединяет сущности: связи образуют ссылку на атрибуты первичного ключа в дочерней сущности и эти атрибуты образуют внешний ключ в дочерней сущности (миграция ключа). Атрибуты внешнего ключа обозначаются символом (FK) после своего имени.

Зависимая сущность может иметь один и тот же ключ из нескольких родительских сущностей. Сущность может также получить один и тот же внешний ключ несколько раз от одного и того же родителя через несколько разных связей. Когда ERwin обнаруживает одно из этих событий, он распознает, что два атрибута одинаковы, и помещает атрибуты внешнего ключа в зависимой сущности только один раз. Это комбинирование или объединение идентичных атрибутов называется унификацией.

Есть случаи, когда унификация нежелательна. Например, когда два атрибута имеют одинаковые имена, но на самом деле они отличаются по смыслу, и необходимо, что бы это отличие отражалось в диаграмме. В этом случае необходимо использовать имена ролей внешнего ключа.

Связи на диаграмме представляются линиями, идущими от одной сущности (таблицы) к другой. Каждой связи присваивается уникальное имя. Связанные таблицы разделяют на родительские и дочерние. Родительские таблицы отображаются прямоугольниками с прямыми углами, дочерние – со скругленными.

После указания всем атрибутам формата данных необходимо созданную логическую модель преобразовать в физическую. Для этого необходимо в **Tools** выбрать **Derive New Model**, где в качестве Target Databases выберите **ODBC/Generic** (для использования в СУБД MySQL) см. Рисунок 9. Наша модель (см Рисунок 4) будет преобразована к виду см.Рисунок 11.

Далее выбрав в меню **Tools/Forward Engineer/Shema Generation**  и задав необходимые настройки, получим в меню Preview код на языке SQL для реализации схемы БД в СУБД MySQL.



Рисунок - Преобразование логической модели в физическую

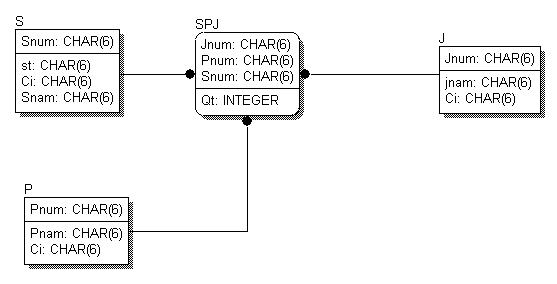


Рисунок - Физическая модель с указанием формата данных.

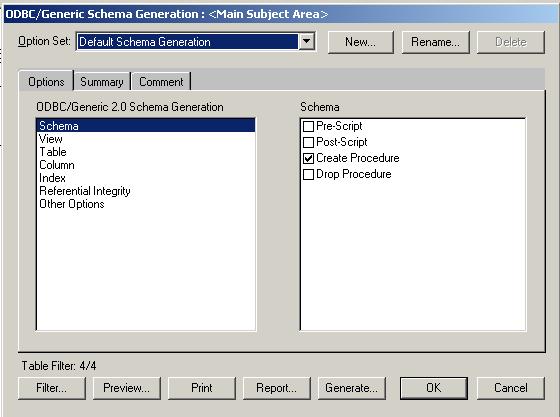


Рисунок - Генерация кода SQL

**Задание**

1. Выполните построение диаграммы с заданными сущностями (прямое моделирование) для заданной предметной области.
2. Задайте атрибуты для каждой определенной сущности. При задании атрибутов используйте домены.
3. Введите связи между сущностями. Присвойте связям уникальные имена.
4. Используя СУБД MYSQL, решите прямую генерацию базы данных для проектируемой информационной.
5. Отчет должен содержать концептуальную модель и физическую базу данных в СУБД MYSQL.

**Контрольные вопросы**

1. В чем состоит различие логического и физического уровней представления моделей данных с помощью ERwin?
2. В чем различие между моделями данных, представленных в форме диаграммы сущность-связь, на основе ключей и в виде полной атрибутивной модели?
3. Какие основные компоненты содержат модели данных, представленные по методологии IDEF1X?

# Лабораторная работа №3. 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](#create) | создание базы данных |
| [Use database](#use) | выбор существующей базы данных |
| close database | закрытие файлов текущей базы данных |
| [drop database](#drop) | удаление базы данных |
| [create table](#create_table) | создание таблицы базы данных |
| [alter table](#alter_table_приложение) | модификация структуры базы данных |
| [drop table](#drop) | удаление таблицы базы данных |
| [insert](#insert) | добавление одной или нескольких строк в таблицу |
| [delete](#delete) | удаление одной или нескольких строк из таблицы |
| [update](#update) | модификация одной или нескольких строк таблицы |
| [LOAD DATA INFILE](#load_data) | 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 1, 2).

Необходимо указать:

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

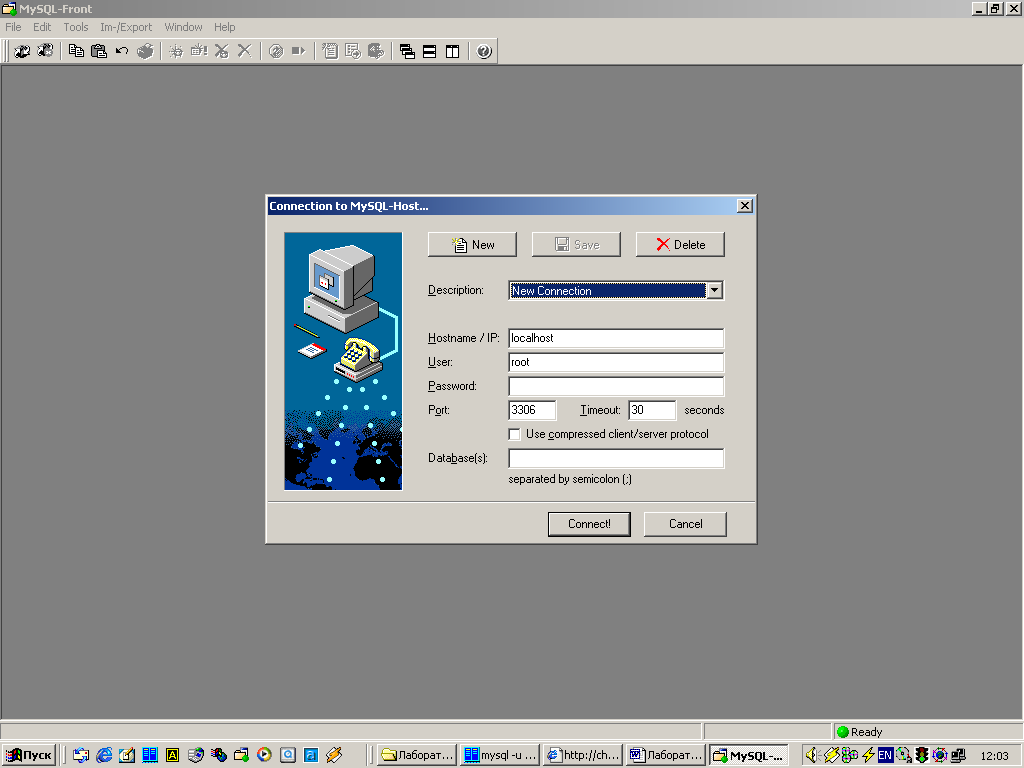


Рисунок - Запуск MySQL-front

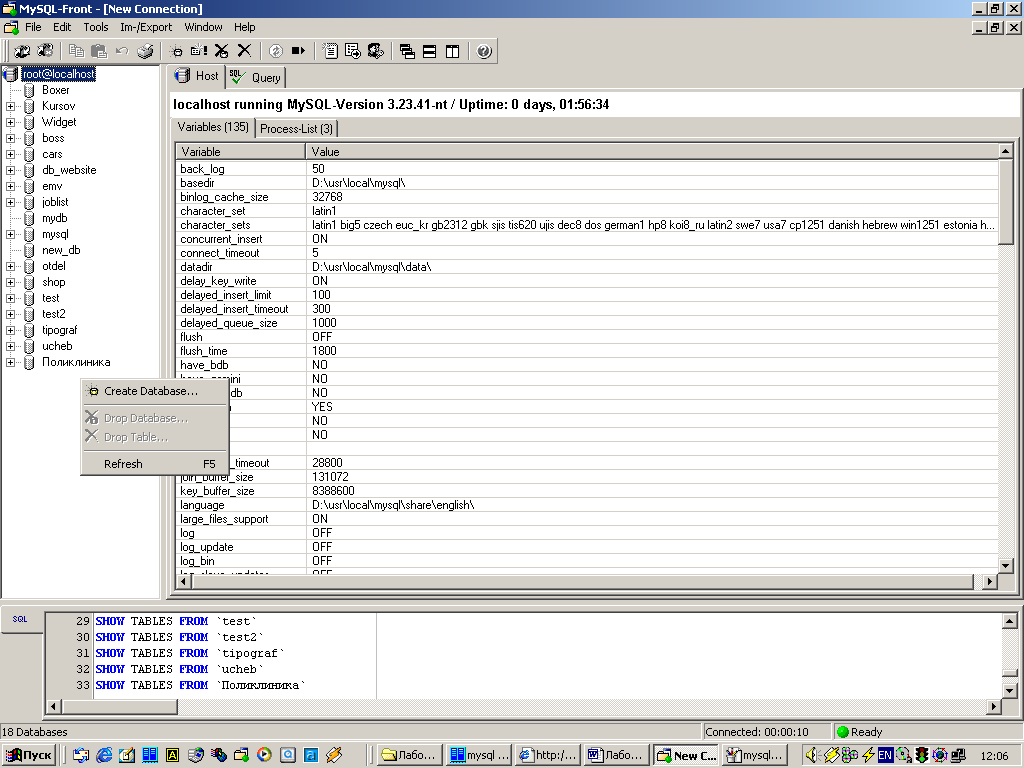


Рисунок - Создание БД в среде MySQL-front

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, как показано на рисунке:

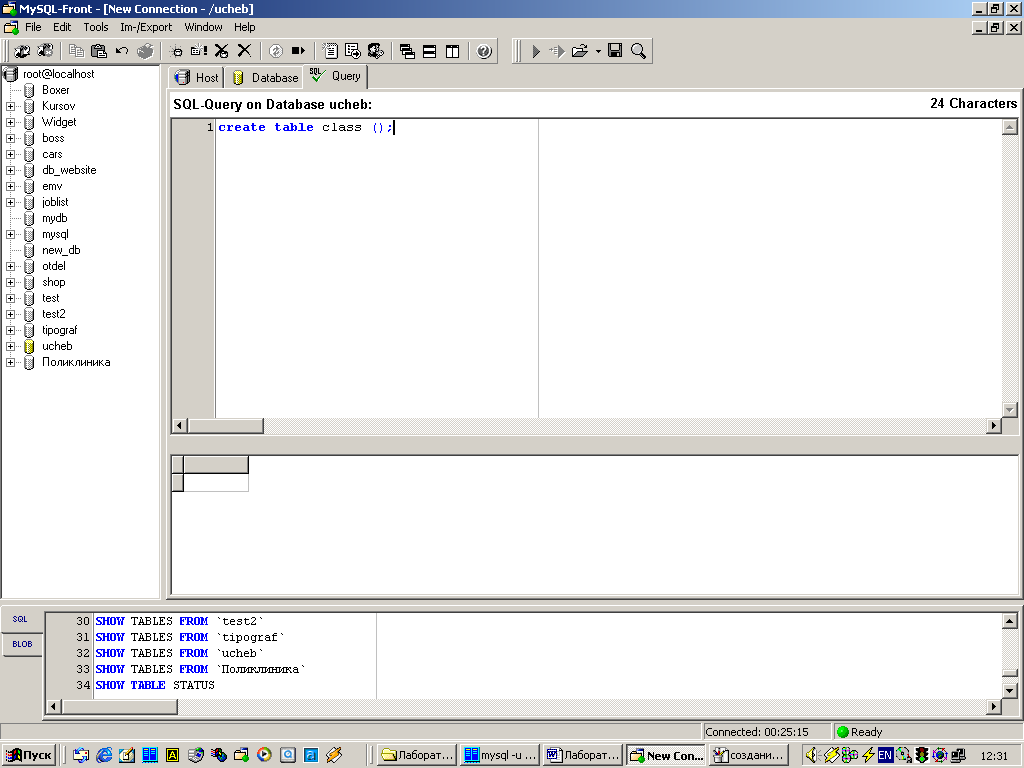


Рисунок - Использование синтаксиса SQL

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

[CREATE TABLE](" \l "create_table_приложение), 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]](#footnote-6)**

указывает имя таблицы, куда будут импортированы данные.

FIELDS TERMINATED BY ';' [[7]](#footnote-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]](#footnote-8)

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

Эти и другие операции можно выполнить также и в программе MySQL-Front.

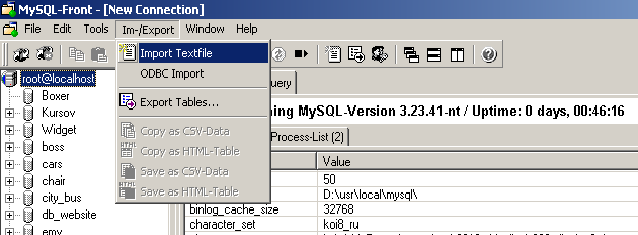


Рисунок - Использование программы MySQL-front для заполнения таблиц данными из файла

Таблица поставщиков (S)

|  |  |  |  |
| --- | --- | --- | --- |
| **Hомеp 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 |
| 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](#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]](#footnote-9). Убедиться в успешности выполненных действий.

7. Выполнить необходимые действия, написав и выполнив соответствующие запросы для модификации таблиц, чтобы структура соответствовала концептуальной модели учебной базы данных (рисунок 5). Убедиться в успешности выполненных действий. При необходимости исправить ошибки.



Рисунок - 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 №4 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: | Hомеp\_поставщика | Фамилия | Рейтинг | Гоpод |
|  | 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 c обратным смыслом. Ключевое слово LIKE соответствует стандарту ANSI.

|  |  |
| --- | --- |
| **СИМВОЛ** | **ЗНАЧЕНИЕ** |
| LIKE |  |
| % | Заменяет последовательность символов |
| - | Заменяет любой одиночный символ |
| \ | Отменяет специальное назначение следующего за ним символа |

**Примеры.**

а) Выбрать список деталей, начинающихся с буквы "Б"[[10]](#footnote-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

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). |
| - С | 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 реляционной алгебры: ограничение, декартово произведение, проекция, пересечение, объединение, разность, соединение?
5. Что такое внешнее соединение?
6. В каких случаях вместо фразы IN можно использовать операцию сравнения?
7. Какие существуют средства группирования в SQL? Как они используются?

# Лабораторная работа №5. 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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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. сетевой машины, где запущен еще один сервер баз данных mysql, и таким образом в случае распределенной базы данных обращение к таблице contracts базы данных salesdb, размещенной на сервере central, будет выглядеть следующим образом: salesdb@central:contracts.

В программе MYSQL-FRONT также существует механизм, обеспечивающий наделение пользователей определенными правами (см. Рисунок 17).

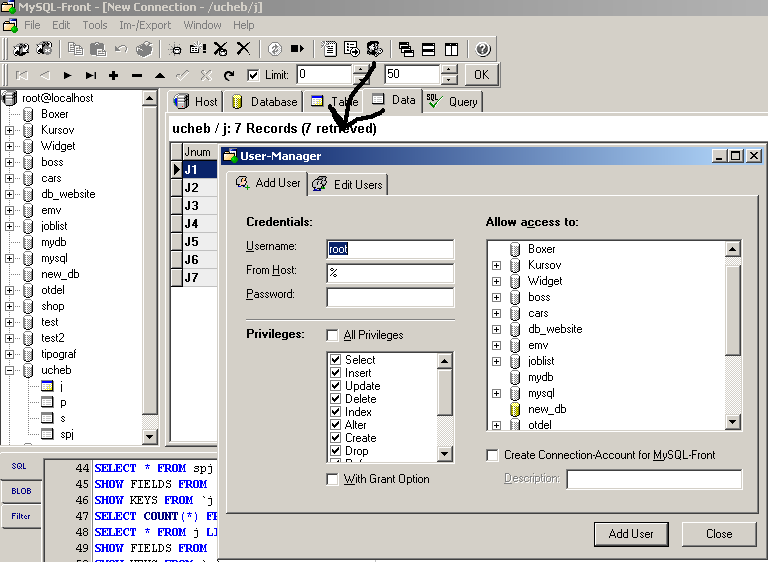


Рисунок - Редактирование прав пользователя

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 данных, и связанные с ними поставки из таблицы SPJ. Убедитесь в невозможности 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 Р1 для какого-либо изделия в количестве, большем среднего объема поставок детали Р1 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?

# Лабораторная работа №6. Представления, хранимые процедуры, функции, триггеры

**Цель работы:** познакомиться с возможностями MySQL по работе с хранимыми процедурами, функциями, триггерами, представлениями.

## Представления

Представления (views) можно сравнить с временными таблицами, наполненными динамически формируемым содержимым.. В настоящей реализации есть две возможности создания представлений: с использованием алгоритма временных таблиц MySQL и с созданием самостоятельной таблицы. Нас интересует именно второй способ (первый был реализован, скорее всего, исходя из соображений совместимости и унификации). Такие представления позволяют значительно снизить объём кода, в котором часто повторялись простые объединения таблиц. К ним (после создания) применимы любые запросы, возвращающие результат в виде набора строк. То есть команды SELECT, UPDATE, DELETE, можно применять так же, как и к реальным таблицам. Важно и то, что посредством представлений можно более гибко распоряжаться правами пользователей базы данных, так как в этом случае есть возможность предоставлять доступ на уровне отдельных записей различных таблиц.

**Создание представлений**

Для создания представлений используется команда CREATE VIEW

**Синтаксис команды CREATE VIEW**

CREATE

[OR REPLACE]

[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]

[DEFINER = { *user* | CURRENT\_USER }]

[SQL SECURITY { DEFINER | INVOKER }]

VIEW *view\_name* [(*column\_list*)]

AS *select\_statement*

[WITH [CASCADED | LOCAL] CHECK OPTION]

Пример создания и работы простейшего представления:

***Create View v as Select column 1 from T***

***Insert into v Values (1)***

***Select \* from v***

**Результат**

**+---------+**

**| column1 |**

**+---------+**

**| 1       |**

**+---------+**

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

Представление может быть создано на основе различных параметров предложения SELECT, при этом можно ссылаться на другие таблицы и представления. Конструкция может использовать оператор UNION и другие подзапросы.

**Синтаксис команды ALTER VIEW**

Для внесения изменений в представление используется команда ALTER VIEW

ALTER

[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]

[DEFINER = { *user* | CURRENT\_USER }]

[SQL SECURITY { DEFINER | INVOKER }]

VIEW *view\_name* [(*column\_list*)]

AS *select\_statement*

[WITH [CASCADED | LOCAL] CHECK OPTION]

**Синтаксис команды DROP VIEW**

Для удаления представления используется команда DROP VIEW

VIEW [IF EXISTS]

*view\_name* [, *view\_name*] ...

[RESTRICT | CASCADE]

ПРИМЕР

mysql> **CREATE TABLE t (qty INT, price INT);**

mysql> **INSERT INTO t VALUES(3, 50);**

mysql> **CREATE VIEW v AS SELECT qty, price, qty\*price AS value FROM t;**

mysql> **SELECT \* FROM v;**

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

| qty | price | value |

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

| 3 | 50 | 150 |

## Хранимые процедуры и функции

В СУБД MySQL появилась возможность создания и хранения функций и процедур. Объявление и работа с процедурами и функциями отличаются в следующем:

* в заголовке функции помимо описания формальных параметров обязательно указывается тип возвращаемого ею результата;
* для возврата функцией значения в точку вызова среди ее операторов должен быть хотя бы один, в котором имени функции или переменной Result присваивается значение результата;
* вызов процедуры выполняется отдельным оператором;
* вызов функции может выполняться там, где допускается ставить выражение, в частности, в правой части оператора присваивания.

Пользовательские функции по функциональности похожи на хранимые процедуры. Разница заключается в том, что возможностей у них меньше (в частности, они должны возвращать только одно значение, например, скалярное или табличное), но их удобнее использовать с точки зрения синтаксиса.

Как процедуры, так и функции могут возвращать значения (в виде набора записей). Различие состоит в том, что функция вызывается из запроса, а процедура из отдельной команды.

На настоящий момент реализация хранимых процедур не поддерживает никаких внешних языков, но (по крайней мере, так заявляется) соответствует стандарту SQL:2003, позволяющему применять условные конструкции, итерации и обработку ошибок.

Пример создания хранимой процедуры в MySQL 5:

***CREATE PROCEDURE p ()***

***LANGUAGE SQL***

***NOT DETERMINISTIC***

***SQL SECURITY DEFINER***

***COMMENT 'A Procedure' <--***

***SELECT CURRENT\_DATE, RAND() FROM t***

В данном случае мы создали процедуру с именем p, которая возвращает текущую дату и псевдослучайное число из таблицы t. Пример ее вызова и возвращаемого результата:

**mysql> call p2()**

**+--------------+-----------------+**

**| CURRENT\_DATE | RAND()          |**

**+--------------+-----------------+**

**| 2005-06-27   | 0.7822275075896 |**

**+--------------+-----------------+**

**1 row in set (0.26 sec)**

**Query OK, 0 rows affected (0.26 sec)**

Чуть более сложный пример создания и использования функции:

***CREATE FUNCTION factorial (n DECIMAL(3,0))***

***RETURNS DECIMAL(20,0)***

***DETERMINISTIC***

***BEGIN***

***DECLARE factorial DECIMAL(20,0) DEFAULT 1;***

***DECLARE counter DECIMAL(3,0);***

***SET counter = n;***

***factorial\_loop: REPEAT***

***SET factorial = factorial \* counter;***

***SET counter = counter - 1;***

***UNTIL counter = 1***

***END REPEAT;***

***RETURN factorial;***

***END***

В приложении:

***INSERT INTO t VALUES (factorial(pi))***

***SELECT s1, factorial (s1) FROM t***

***UPDATE t SET s1 = factorial(s1)***

***WHERE factorial(s1) < 5***

Разумеется эффективность применения хранимых процедур существенно возрастает при вызове их с параметрами (аргументами). Ниже дан пример процедуры с обработкой переданных ей параметров:

***CREATE PROCEDURE p1 (IN parameter1 INT)***

***BEGIN***

***DECLARE variable1 INT;***

***SET variable1 = parameter1 + 1;***

***IF variable1 = 0 THEN***

***INSERT INTO t VALUES (17);***

***END IF;***

***IF parameter1 = 0 THEN***

***UPDATE t SET s1 = s1 + 1; <--***

***ELSE***

***UPDATE t SET s1 = s1 + 2;***

***END IF;***

***END;***

Вызов процедуры теперь будет таким:

***mysql> CALL p2(0) // Query OK, 2 rows affected (0.28 sec)***

и в результате запроса мы получим:

**mysql> SELECT \* FROM t**

**+----+**

**| s1 |**

**+----+**

**| 6  |**

**| 6  |**

**+------+**

**2 rows in set (0.01 sec)**

Кроме условных, возможны и любые циклические конструкции:

***CREATE PROCEDURE p3 ()***

***BEGIN***

***DECLARE v INT;***

***SET v = 0;***

***WHILE v < 5 DO***

***INSERT INTO t VALUES (v);***

***SET v = v + 1;***

***END WHILE;***

***END;***

Вызов процедуры:

**mysql> CALL p3()**

**+------+**

**| s1   |**

**+------+**

**…………**

**|    0 |**

**|    1 |**

**|    2 |**

**|    3 |**

**|    4 |**

**+------+**

**Query OK, 1 row affected (0.00 sec)**

Также применимы итерации, переходы, словом, всё, что предполагает стандарт.

Внутри функций и хранимых процедур осуществлена реализация курсоров, но, к сожалению, она пока ограничена (ASESITIVE, READ ONLY и NONSCROLL):

***CREATE PROCEDURE p25 (OUT return\_val INT)***

***BEGIN***

***DECLARE a,b INT;***

***DECLARE cur\_1 CURSOR FOR SELECT s1 FROM t;***

***DECLARE CONTINUE HANDLER FOR NOT FOUND***

***SET b = 1;***

***OPEN cur\_1;***

***REPEAT***

***FETCH cur\_1 INTO a;***

***UNTIL b = 1***

***END REPEAT;***

***CLOSE cur\_1;***

***SET return\_val = a;***

***END;***

**Создание процедур и функций**

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*

*func\_parameter*:

*param\_name* *type*

*type*:

*Any valid MySQL data type*

*characteristic*:

LANGUAGE SQL

| [NOT] DETERMINISTIC

| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

| COMMENT '*string*'

*routine\_body*:

**Внесение изменений**

ALTER {PROCEDURE | FUNCTION} *sp\_name* [*characteristic* ...]

*characteristic*:

{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }

| SQL SECURITY { DEFINER | INVOKER }

| COMMENT '*string*'

**Удаление процедур и функций**

DROP {PROCEDURE | FUNCTION} [IF EXISTS] *sp\_name*

**Вызов процедур и функций**

CALL *sp\_name*([*parameter*[,...]])

CALL *sp\_name*[()]

Оператор CALL позволяет вызвать ранее определенную процедуру.

**Пример1**

***CREATE PROCEDURE p1 (OUT ver\_param VARCHAR(25), INOUT incr\_param INT)***

***BEGIN***

***# Set value of OUT parameter***

***SELECT VERSION() INTO ver\_param;***

***# Increment value of INOUT parameter***

***SET incr\_param = incr\_param + 1;***

***END;***

Перед вызовом процедуры инициализируйте переменную указанные в параметрах INOUT . После вызова процедуры значения будут установлены или изменены.

mysql> **SET @increment = 10;**

mysql> **CALL p(@version, @increment);**

mysql> **SELECT @version, @increment;**

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

| @version | @increment |

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

| 5.1.12-beta-log | 11 |

**Пример2**

***CREATE PROCEDURE `p2`(IN param1 CHAR(2) )***

***NOT DETERMINISTIC***

***SQL SECURITY DEFINER***

***COMMENT ''***

***BEGIN***

***select \* from s where snum=param1;***

***END;***

Вызов процедуры

call p2 ('S1')

**Пример3**

***CREATE PROCEDURE `My\_proc2`(IN param1 CHAR(2) )***

***BEGIN /\* start of block \*/***

***DECLARE variable1 CHAR(10); /\* variables \*/***

***IF param1 = 17 THEN /\* start of IF \*/***

***SET variable1 = 'birds'; /\* assignment \*/***

***ELSE***

***SET variable1 = 'beasts'; /\* assignment \*/***

***END IF; /\* end of IF \*/***

***select variable1; /\* statement \*/***

***END***

Вызов процедуры

call p3 (10)

## Триггеры

Три́ггер (англ. trigger) — это хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено наступлением определенного события (действием) — по сути добавлением INSERT или удалением DELETE строки в заданной таблице, или модификаци UPDATE данных в определенном столбце заданной таблицы реляционной базы данных. Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики. Триггер запускается сервером автоматически при попытке изменения данных в таблице, с которой он связан. Все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции. Момент запуска триггера определяется с помощью ключевых слов BEFORE (триггер запускается до выполнения связанного с ним события; например, до добавления записи) или AFTER (после события). В случае, если триггер вызывается до события, он может внести изменения в модифицируемую событием запись (конечно, при условии, что событие — не удаление записи). Некоторые СУБД накладывают ограничения на операторы, которые могут быть использованы в триггере (например, может быть запрещено вносить изменения в таблицу, на которой «висит» триггер, и т. п.)

Кроме того, триггеры могут быть привязаны не к таблице, а к представлению (VIEW). В этом случае с их помощью реализуется механизм «обновляемого представления». В этом случае ключевые слова BEFORE и AFTER влияют лишь на последовательность вызова триггеров, так как собственно событие (удаление, вставка или обновление) не происходит.

***CREATE***

***[DEFINER = { user | CURRENT\_USER }]***

***TRIGGER trigger\_name trigger\_time trigger\_event***

***ON tbl\_name FOR EACH ROW trigger\_stmt***

Пример создания и работы триггера:

***CREATE TABLE t22 (s1 INTEGER)***

***CREATE TRIGGER t22\_bi***

***BEFORE INSERT ON t22***

***FOR EACH ROW***

***BEGIN***

***SET @x = ‘Trigger was activated!’;***

***SET NEW.s1 = 55;***

***END;***

После этого при выполнении запросов получим:

**mysql> INSERT INTO t22 VALUES (1)**

**mysql> SELECT @x, t22.\* FROM t22 // вызывается триггер**

**+------------------------+------+**

**| @x                     | s1   |**

**+------------------------+------+**

**| Trigger was activated! | 55   |**

**+------------------------+------+**

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

## Словарь данных

Иметь доступ к значениям метаданных – совершенно необходимое требование к современной СУБД. Ранее такая возможность в MySQL достигалась различными SHOW-командами, но такой подход имеет очевидные недостатки. Эти команды нельзя использовать в простых запросах с соединениями, и, что существенно, они не соответствовали стандартам, будучи специфичными для MySQL.

В новой версии СУБД появилась новая служебная база данных – INFORMATION\_SCHEMA. Её наличие продиктовано тем же стандартом SQL:2003, и именно она решает задачу реализации словаря данных (data dictionary). INFORMATION\_SCHEMA содержит таблицы, описывающие состояние и параметры сервера, в том числе определения и сущности таблиц. Это виртуальная база данных – физически (в виде файлов на диске) она не существует, вся информация динамически предоставляется сервером. Пример использования этой таблицы:

**mysql> SELECT table\_name, table\_type, engine**

**-> FROM INFORMATION\_SCHEMA.tables**

**-> WHERE table\_schema = ‘tp’**

**-> ORDER BY table\_type ASC, table\_name DESC;**

**+------------+------------+--------+**

**| table\_name | table\_type | engine |**

**+------------+------------+--------+**

**| t2         | BASE TABLE | MyISAM |**

**| t1         | BASE TABLE | InnoDB |**

**| v1         | VIEW       | NULL   |**

**+------------+------------+--------+**

Другой пример работы со словарём данных – просмотр привелегий:

**mysql> SELECT \* FROM**

**-> INFORMATION\_SCHEMA.COLUMN\_PRIVILEGES\G**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**GRANTEE: 'peter'@'%'**

**TABLE\_CATALOG: NULL**

**TABLE\_SCHEMA: tp**

**TABLE\_NAME: t1**

**COLUMN\_NAME: col1**

**PRIVILEGE\_TYPE: UPDATE**

**IS\_GRANTABLE: NO**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**GRANTEE: 'trudy'@'%'**

**TABLE\_CATALOG: NULL**

**TABLE\_SCHEMA: tp**

**TABLE\_NAME: t2**

**COLUMN\_NAME: col1**

**PRIVILEGE\_TYPE: SELECT**

**IS\_GRANTABLE: YES**

**Объявление переменных**

Объявление. DECLARE Local Variables

Следующая команда позволяет объявлять локальные переменные, содержит возможность задания значения по умолчанию. Переменная может быть объявлена как выражения, не обязательна константа. Если значение по умолчанию не определено то равно NULL.

DECLARE *var\_name*[,...] *type* [DEFAULT *value*]

Присваивание Variable SET Statement

SET *var\_name* = *expr* [, *var\_name* = *expr*] ...

SELECT ... INTO Statement

Оператор SELECT может перенаправить результат в переменные. Таким образом может быть преобразована только одна строка.

ПРИМЕР

SELECT *col\_name*[,...] INTO *var\_name*[,...] *table\_expr*

SELECT id,data INTO x,y FROM test.t1 LIMIT 1;

### 

**Условия и ограничения**

**Объявление условий**

DECLARE *condition\_name* CONDITION FOR *condition\_value*

*condition\_value*:

SQLSTATE [VALUE] *sqlstate\_value*

| *mysql\_error\_code*

**Объявление ограничений**

DECLARE *handler\_type* HANDLER FOR *condition\_value*[,...] *statement*

*handler\_type*:

CONTINUE

| EXIT

| UNDO

*condition\_value*:

SQLSTATE [VALUE] *sqlstate\_value*

| *condition\_name*

| SQLWARNING

| NOT FOUND

| SQLEXCEPTION

| *mysql\_error\_code*

**Пример**

mysql> **CREATE TABLE test.t (s1 int,primary key (s1));**

Query OK, 0 rows affected (0.00 sec)

mysql> **delimiter //**

mysql> **CREATE PROCEDURE handlerdemo ()**

-> **BEGIN**

-> **DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;**

-> **SET @x = 1;**

-> **INSERT INTO test.t VALUES (1);**

-> **SET @x = 2;**

-> **INSERT INTO test.t VALUES (1);**

-> **SET @x = 3;**

-> **END;**

-> **//**

Query OK, 0 rows affected (0.00 sec)

mysql> **CALL handlerdemo()//**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT @x//**

+------+

| @x |

+------+

| 3 |

+------+

1 row in set (0.00 sec)

Если вы хотите игнорировать условие вы должны сгенерировать ограничение и ассоциировать его с пустым блоком .

DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;

**Пример**

CREATE PROCEDURE p ()

BEGIN

DECLARE i INT DEFAULT 3;

retry:

REPEAT

BEGIN

DECLARE CONTINUE HANDLER FOR SQLWARNING

BEGIN

ITERATE retry; # illegal

END;

END;

IF i < 0 THEN

LEAVE retry; # legal

END IF;

SET i = i - 1;

UNTIL FALSE END REPEAT;

END;

## Курсоры

Курсор — в некоторых реализациях языка программирования SQL (Oracle, Microsoft SQL Server) — получаемый при выполнении запроса результирующий набор и связанный с ним указатель текущей записи.

Курсор может возвращать одну строку, несколько строк или ни одной строки. Для запросов, возвращающих более одной строки, можно использовать только явный курсор. Для повторного создания результирующего набора для других значений параметров курсор следует закрыть, а затем повторно открыть.

Курсор может быть объявлен в секциях объявлений любого блока PL/SQL, подпрограммы или пакета.

Операторы управления явным курсором

* Оператор CURSOR выполняет объявление явного курсора.
* Оператор OPEN открывает курсор, создавая новый результирующий набор на базе указанного запроса.
* Оператор FETCH выполняет последовательное извлечение строк из результирующего набора от начала до конца.
* Оператор CLOSE закрывает курсор и освобождает занимаемые им ресурсы

Курсоры поддерживают хранимые процедуры и функции. Сейчас курсоры имеют три свойства:

* Asensitive: The server may or may not make a copy of its result table
* Read only: Not updatable
* Non-scrollable: Can be traversed only in one direction and cannot skip rows

Курсоры должны быть объявлены перед объявлением ограничений. Переменные и условия должны быть объявлены перед курсором.

**Объявление курсоров**

Оператор объявления курсора. М программе можно объявлять несколько курсоров, каждый курсор в блоке должен иметь уникальное имя.

DECLARE *cursor\_name* CURSOR FOR *select\_statement*

**Условие открытия  Cursor OPEN Statement**

Оператор открывает ранее объявленный курсор.

OPEN *cursor\_name*

**Выполнение курсора  Cursor FETCH Statement**

FETCH *cursor\_name* INTO *var\_name* [, *var\_name*] ...

**Условия закрытия  Cursor CLOSE Statement**

CLOSE *cursor\_name*

**Пример**

CREATE PROCEDURE curdemo()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE a CHAR(16);

DECLARE b,c INT;

DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;

DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN cur1;

OPEN cur2;

REPEAT

FETCH cur1 INTO a, b;

FETCH cur2 INTO c;

IF NOT done THEN

IF b < c THEN

INSERT INTO test.t3 VALUES (a,b);

ELSE

INSERT INTO test.t3 VALUES (a,c);

END IF;

END IF;

UNTIL done END REPEAT;

CLOSE cur1;

CLOSE cur2;

END

**Задание**

**Представления**

1. Составить представление, возвращающее объем поставок деталей для изделий за заданный календарный месяц
2. Добавить столбец стоимость детали в таблицу SPJ. Создать соответствующее представление (наименование поставщика, наименование детали, наименование изделия, стоимость детали, количество, стоимость поставки.
3. Добавить столбец стоимость детали в таблицу P. Создать представление отражающее стоимость поставки.

**Процедуры**

1. Составить процедуру, отражающую состав изделия (детали изделия).
2. Составить процедуру, возвращающую расчетную стоимость изделия, учитывая, что для изделия требуется K деталей каждого требуемого наименования (см. табл 1).
3. Составить процедуру , отражающую вес изделия (п4) учитывая что для изделия требуется K деталей каждого требуемого наименования (см. табл 1).
4. С помощью условных операторов разделить всех поставщиков на три категории по количеству поставляемых деталей (ABC анализ) 20 40 60 %
5. Тоже, но по стоимости поставки
6. Определить оптимального поставщика для изделия (см табл 1) для производства максимального количества изделий за период
7. Определить оптимального поставщика для изделия (см табл 1) для производства максимального количества изделий по минимальной стоимости
8. При условии, что поставщик может поставлять не более одной поставки в неделю, а максимальное количество деталей в поставке не выше среднего за период

**Функции**

1. С помощью функций получить таблицу, отражающую информацию о перечне изделий, для которого выполняется поставка

|  |  |
| --- | --- |
| S1 | J1 J2 J4 |
| S2 | J5 |

1. Тоже, но с наименованиями изделий.
2. С помощью функций получить таблицу, отражающую информацию о перечне деталей из которых состоит дневная поставка
3. Тоже , но с наименованиями деталей
4. Получить наименование поставщика поставляемого самое большое количество деталей
5. Получить наименование поставщика поставляемого самое большое количество деталей, для какого либо изделия

**Работа с текстовым файлом**

1. Создать текстовый файл, содержащий информацию о поставщике, поставившего за последний месяц деталей на большую сумму и меньшего веса.

**Курсоры**

1. При заполнении поставки поле дата всегда заполнять текущей датой.

Таблица -Варианты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| задание | Вар 1 | Вар 2 | Вар 3 | Вар 4 | Вар 5 |
| 1,9,10 | J1 | J2 | J3 | J4 | J5 |
| ЯНВАРЬ | ФЕФРАЛЬ | МАРТ | АПРЕЛЬ | МАЙ |
| 5,6 | 15 | 20 | 25 | 30 | 35 |
| 18 | Max;Min | >AVG;Min | MIN;>AVG | >AVG;>AVG | Max;Max |

**ПРИЛОЖЕНИЕ**

**Синтаксис оператора *CREATE TABLE***

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name [(create\_definition,...)]

[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 равнозначна установке его в NULL, поскольку 0 для TIMESTAMP является допустимой величиной.

Величина DEFAULT должна быть константой, она не может быть функцией или выражением. Если для данного столбца не задается никакой величины DEFAULT, то MySQL автоматически назначает ее. Если столбец может принимать NULL как допустимую величину, 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 равно 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 попытке 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 некоторое приложение запрашивает его, то MySQL может превратить в PRIMARY KEY первый 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. Необходимо использовать синтаксис PRIMARY KEY(index\_col\_name, ...).

Если ключ PRIMARY или UNIQUE состоит только из одного столбца и он принадлежит 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 уникальным. Имена индексов для таблицы можно увидеть, используя SHOW INDEX FROM tbl\_name. SHOW Syntax.

С помощью выражения col\_name(length) можно указать индекс, для которого 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 указывать 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 делают. 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 таблица. 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 только строки, но не числа. При использовании PACK\_KEYS=1 числа тоже будут уплотняться. 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 умолчанию. 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 DEFAULT}

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\_столбца 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 пытается 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 добавлении 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 неуникальные 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 автоматически 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 оператора 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 таблицы.

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. Например:

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\_BUFFER\_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 | formula} [ASC | DESC], ...]

[HAVING where\_definition]

[ORDER BY {unsigned\_integer | col\_name | formula} [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) |
| - С | 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 |

Учебное издание

ПОНОМАРЕВ Алексей Анатольевич

Часть 1

# *СОЗДАНИЕ СТРУКТУРЫ БД И МАНИПУЛИРОВАНИЕ ДАННЫМИ В СУБД MYSQL*

Методические указания к выполнению лабораторных работ

по курсу «Базы данных» для студентов III курса специальностей 230105 «Программное обеспечение вычислительной техники и автоматизированных систем». 230201 «Информационные системы и технологии».

Научный редактор

доктор технических наук,

профессор *Г.П. Цапко*

|  |  |  |
| --- | --- | --- |
| Подписано к печати 26.11.2009. Формат 60х84/16. Бумага «Снегурочка».  Печать Xerox. Усл. печ. л. 000. Уч.-изд. л. 000.  Заказ ХХХ. Тираж ХХХ экз. | | |
|  | Томский политехнический университет  Система менеджмента качества  Томского политехнического университета сертифицирована  NATIONAL QUALITY ASSURANCE по стандарту ISO 9001:2000 |  |
| . 634050, г. Томск, пр. Ленина, 30. | | |

1. Эта и все, описанные ниже команды добавлены начиная с версии 5.12 [↑](#footnote-ref-1)
2. ***Атрибуты***  ***ssl\_cipher, x509\_issuer, x509\_subject обязательны для заполнения для версии сервера 5.12*** [↑](#footnote-ref-2)
3. Для случая, если работаете на том же компьютере где запущен сервер БД [↑](#footnote-ref-3)
4. Даны для справки, для текущей версии сервера может быть существенно расширены [↑](#footnote-ref-4)
5. Перечень типов данных, поддерживаемых СУБД необходимо уточнить у производителя [↑](#footnote-ref-5)
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. [↑](#footnote-ref-6)
7. В случае использования табулятора \t [↑](#footnote-ref-7)
8. В случае Enter \r\n [↑](#footnote-ref-8)
9. См утилиту Mysqldump [↑](#footnote-ref-9)
10. Примечание. Корректно работает только при задании кодировки по умолчанию. Задается в разделе MYSQLD default\_character\_set=win1251 [↑](#footnote-ref-10)