МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)

Колледж экономики, управления и права

Методические указания

по организации практических занятий

по ПМ.02 Участие в разработке информационных систем

МДК.02.01 Информационные технологии и платформы разработки информационных систем

6 семестр

Специальность
09.02.04 Информационные системы (по отраслям)
6 семестр

Ростов-на-Дону
2020

Методические указания по ПМ.02 Участие в разработке информационных систем МДК.02.01 Информационные технологии и платформы разработки информационных систем разработаны с учетом ФГОС среднего профессионального образования специальности 09.02.04 Информационные системы (по отраслям), предназначены для студентов и преподавателей колледжа.

Методические указания содержат описание: программного продукта, подключения к базе данных, создания базы данных и редактирования данных.

Составитель (автор): С.В.Шинакова, преподаватель колледжа ЭУП

Рассмотрены на заседании предметной (цикловой) комиссии специальности 09.02.04 Информационные системы (по отраслям)

Протокол № 1 от 31 августа 2020 г

Председатель П(Ц)К специальности _____ С.В.Шинакова
личная подпись

и одобрены решением учебно-методического совета колледжа.

Протокол № 1 от 31 августа 2020 г

Председатель учебно-методического совета колледжа
_____ С.В.Шинакова
личная подпись

Рекомендованы к практическому применению в образовательном процессе.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4
Практическое занятие №1 Работа в CRM-системе SupaSoft Free Lite 5
Практическое занятие № 2 Создание базы данных, основы работы с таблицами 5
Практическое занятие № 3 Работа с таблицами 10
Практическое занятие № 4 Работа с таблицами 12
Практическое занятие № 5 18
Часть 1 Логические операторы 18
Часть 2 Команды обработки данных 22
Практическое занятие № 6 28
Часть 1 Математические функции 28
Часть 2 Работа с датой и временем 47
Практическое занятие № 7 Хранимые процедуры 51
Практическое занятие № 8 Хранимые процедуры 60
Практическое занятие № 9 Создание триггеров 70
Практическое занятие №10 Основы работы с dbForge 72
Практическое занятие № 11 75
Часть 1 Работа с таблицами 75
Часть 2 Работа с запросами dbForge 78
Часть 3 Формирование триггеров 85
Часть 4 Резервное копирование / восстановление 86
Практическое занятие № 12-22 Подготовка курсовой работы 87
Практическое занятие № 23-25 Разработка информационной системы средствами Visual Studio и MySql 88
Практическое занятие № 26-27 Разработка индивидуальной информационной системы 101
Список использованных источников 101

ВВЕДЕНИЕ

Проектировать базу данных можно с помощью SQL-команд, используя непосредственно инструменты MySQL, но этот вариант является достаточно кропотливым и трудоемким, потому существует достаточно много программных продуктов, значительно облегчающих эту непростую работу.

Одним из таких гибких профессиональных инструментов для разработчиков и пользователей MySQL является программа dbForge Studio for SQL от компании Devart. С ее помощью автоматизируются рутинные задачи разработки и администрирования MySQL: проектирование базы данных, ввод и редактирование данных таблиц, создание и выполнение SQL-скриптов, запросов и триггеров и др. Программа является бесплатной для частного некоммерческого использования и для учебных заведений.

Программа имеет русскую локализацию, однако ее справочное пособие (кстати, довольно неплохое) имеется только на английском языке.

Практическое занятие №1 Работа в CRM-системе SupaSoft Free Lite

Цель занятия: формирование навыка работы с CRM-системой

Этапы выполнения работы:

1. Запустить SupaSoft Free Lite.
2. Ознакомиться с интерфейсом программы.
3. Заполнить справочные таблицы.
4. Создать отчеты.
5. Описать этапы работы, приложить скриншоты.
6. Защитить выполненную работу.

Практическое занятие № 2 Создание базы данных, основы работы с таблицами

Цель занятия: формирование навыка работы с таблицами и их заполнение

Этапы выполнения работы:

1. Запустить MySql.
2. Создать базу данных employees.
3. Продемонстрировать полученные таблицы преподавателю, защитить выполненную работу.

ХОД РАБОТЫ

Теоретические сведения

1 Создание базы данных

Синтаксис команды CREATE DATABASE имеет вид:

```
CREATE DATABASE [IF NOT EXISTS] имя_базы_данных
[спецификация_create[,спецификация_create]...]
```

Команда CREATE DATABASE создает базу данных с указанным именем. Для использования команды необходимо иметь привилегию CREATE для базы данных. Если база данных с таким именем существует, генерируется ошибка.

```
спецификация_create:
    [DEFAULT] CHARACTER SET имя_набора_символов
    [DEFAULT] COLLATE имя_порядка_сопоставления
```

Опция спецификация_create может указываться для определения характеристик базы данных. Характеристики базы данных сохраняются в файле db.opt, расположенном в каталоге данных. Конструкция CHARACTER SET определяет набор символов для базы данных по умолчанию. Конструкция COLLATION задает порядок сопоставления по умолчанию.

Базы данных в MySQL реализованы в виде каталогов, которые содержат файлы, соответствующие таблицам базы данных. Поскольку изначально в базе нет никаких таблиц, оператор CREATE DATABASE только создает подкаталог в каталоге данных MySQL.

2 Создание таблиц

Общий формат инструкции CREATE TABLE таков:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] имя
```

[(спецификация, ...)]
[опция, ...]
[[IGNORE | REPLACE] запрос]

Флаг TEMPORARY задает создание временной таблицы, существующей в течение текущего сеанса. По завершении сеанса таблица удаляется. Временным таблицам можно присваивать имена других таблиц, делая последние временно недоступными. Спецификатор IF NOT EXIST подавляет вывод сообщений об ошибках в случае, если таблица с указанным именем уже существует. Имени таблицы может предшествовать имя базы данных, отделенное точкой. Если это не сделано, таблица будет создана в базе данных, которая установлена по умолчанию.

Чтобы задать имя таблицы с пробелами, необходимо заключить его в обратные кавычки, например 'courses list'. То же самое нужно будет делать во всех ссылках на таблицу, поскольку пробелы используются для разделения идентификаторов.

Разрешается создавать таблицы без столбцов, однако в большинстве случаев спецификация хотя бы одного столбца все же присутствует. Спецификации столбцов и индексов приводятся в круглых скобках и разделяются запятыми. Формат спецификации следующий:

имя тип
[NOT NULL | NULL]
[DEFAULT значение]
[AUTO_INCREMENT]
[KEY]
[ссылка]

Спецификация типа включает название типа и его размерность. По умолчанию столбцы принимают значения NULL. Спецификатор NOT NULL запрещает подобное поведение.

У любого столбца есть значение по умолчанию. Если оно не указано, программа MySQL выберет его самостоятельно. Для столбцов, принимающих значения NULL, значением по умолчанию будет NULL, для строковых столбцов — пустая строка, для численных столбцов — нуль. Изменить эту установку позволяет предложение DEFAULT.

Поля-счетчики, создаваемые с помощью флага AUTO_INCREMENT, игнорируют значения по умолчанию, так как в них записываются порядковые номера. Тип счетчика должен быть беззнаковым целым. В таблице может присутствовать лишь одно поле-счетчик. Им не обязательно является первичный ключ.

3 Удаление таблиц

Инструкция DROP TABLE имеет следующий синтаксис:

DROP TABLE [IF EXISTS] таблица [RESTRICT | CASCADE]

Спецификация IF EXISTS подавляет вывод сообщения об ошибке, выдаваемого в случае, если заданная таблица не существует. Можно указывать несколько имен таблиц, разделяя их запятыми.

Флаги RESTRICT и CASCADE предназначены для выполнения сценариев, созданных в других СУБД.

4 Заполнение таблиц данными

Для этого используется оператор *INSERT*.
Синтаксис можно использовать нескольких видов.

Первый вариант используется для внесения данных во все поля таблицы:

INSERT INTO имя_таблицы VALUES
('значение_первого_столбца','значение_второго_столбца',…,'значение_последнего_столбца');

Второй вариант используется для внесения данных в некоторые поля таблицы:

INSERT INTO имя_таблицы ('имя_столбца', 'имя_столбца') VALUES ('значение_первого_столбца','значение_второго_столбца');

Третий вариант используется для внесения нескольких записей в таблицу:

Insert [Low_Priority | Delayed] [Ignore]
[into] Имя_таблицы [(поле1, поле2, …)]
Values (знач1, знач2,…),
(знач1, знач2,…),
…
;

Если указывается ключевое слово LOW_PRIORITY, то выполнение данной команды INSERT будет задержано до тех пор, пока другие клиенты не завершат чтение этой таблицы. В этом случае данный клиент должен ожидать, пока данная команда вставки не будет завершена, что в случае интенсивного использования таблицы может потребовать значительного времени. В противоположность этому команда INSERT DELAYED позволяет данному клиенту продолжать операцию сразу же.

Следует отметить, что указатель LOW_PRIORITY обычно не используется с таблицами MyISAM, поскольку при его указании становятся невозможными параллельные вставки.

Если в команде INSERT со строками, имеющими много значений, указывается ключевое слово IGNORE, то все строки, имеющие дублирующиеся ключи PRIMARY или UNIQUE в этой таблице, будут проигнорированы и не будут внесены. Если не указывать IGNORE, то данная операция вставки прекращается при обнаружении строки, имеющей дублирующееся значение существующего ключа. Количество строк, внесенных в данную таблицу, можно определить при помощи функции C APImysql_info().

INSERT [LOW_PRIORITY] [IGNORE] [INTO] tbl_name [(column list)] SELECT ...

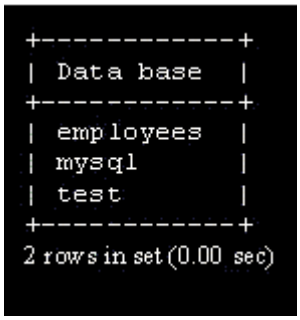
Команда INSERT ... SELECT обеспечивает возможность быстрого внесения большого количества строк в таблицу из одной или более таблиц.

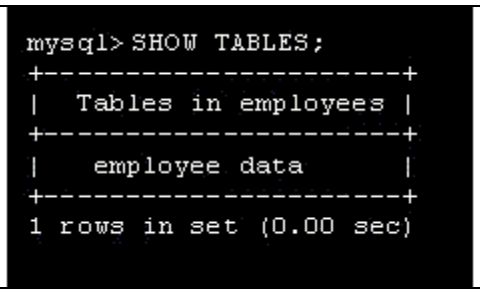
```
INSERT INTO tblTemp2 (fldID) SELECT tblTemp1.fldOrder_ID
    FROM tblTemp1
    WHERE tblTemp1.fldOrder_ID > 100;
```

5 Модификация таблицы

`Alter Table имя_таблицы Modify имя-поля новый_тип;` - изменение типа существующего поля;
`Alter Table имя_таблицы Add имя-поля тип;` - добавление поля в таблицу

Практические задания
Создание базы данных в Windows

1.	Запустите сервер MySQL	
2.	Затем вызовите программу клиента mysql	
3.	Введите команду:	create database employees;
4.	Сервер MySQL должен ответить примерно как на рис.  Это означает, что была успешно создана база данных.	
5.	Теперь посмотрим, сколько баз данных имеется в системе. Выполните следующую команду.	show databases;
6.	Сервер ответит списком баз данных, как показано на рис.  Здесь показаны три базы данных, две были созданы MySQL во время установки и вновь созданная база данных employees.	
7.	Чтобы вернуться снова к приглашению DOS, введите команду quit в приглашении mysql.	
8.	Создать таблицу employee_data	`CREATE TABLE employee_data` `(` `emp_id int unsigned not null auto_increment` `primary key,` `f_name varchar(20),` `l_name varchar(20),` `title varchar(30),` `age int,` `yos int,` `salary int,` `perks int,` `email varchar(60)` `);`
9.	Описание ограничений: – int: определяет тип столбца как целое число; – unsigned: определяет, что число будет без знака (положительное целое); – not null: определяет, что значение не может быть null (пустым); то есть каждая	

	строка в этом столбце должна иметь значение; – auto_increment: когда MySQL встречается со столбцом с атрибутом auto_increment, то генерируется новое значение, которое на единицу больше, чем наибольшее значение в столбце. Поэтому мы не должны задавать для этого столбца значения, MySQL генерирует их самостоятельно. Из этого также следует, что каждое значение в этом столбце будет уникальным; – primary key: помогает при индексировании столбца, что ускоряет поиск значений. Каждое значение должно быть уникально. Ключевой столбец необходим для того, чтобы исключить возможность совпадения данных. Например, два сотрудника могут иметь одно и то же имя, и тогда встанет проблема – как различать этих сотрудников, если не задать им уникальные идентификационные номера. Если имеется столбец с уникальными значениями, то можно легко различить две записи. Лучше всего поручить присваивание уникальных значений самой системе MySQL.	
10.	Для работы с БД, необходимо её "активировать" или "выбрать". В приглашении mysql выполните команду:	`SELECT DATABASE();`
11.	Просмотр таблиц в базе	
12.	Заполнить таблицу данными	`INSERT INTO employee_data` `(f_name, l_name, title, age, yos, salary, perks, email)` `values` `("Сергей", "Степаненко", "директор", 35, 7, 100000,50000, "serg777@mail.ru");` `Внести 5 записей.`
13.	Измените тип поля title на Next	Alter Table имя_таблицы Modify имя-поля новый_тип
14.	Добавьте в таблицу поле Пароль	Alter Table имя_таблицы Add имя-поля тип

Создать следующие запросы:

1. Напишите оператор для записи следующих данных в таблицу employee_data

```
Имя: Николай
Фамилия: Крячков
Должность: Программист
Возраст: 33
Стаж работы в компании: 5
Зарплата: 60000
Надбавки: 20000
email: nik161@yandex.ru
```

2. Приведите две формы оператора SELECT, которые будут выводить все данные из таблицы employee_data.

3. Как извлечь данные столбцов f_name, email из таблицы employee_data?

4Напишите оператор для вывода данных из столбцов salary, perks и yos таблицы employee_data.

5 Как узнать число строк в таблице с помощью оператора SELECT?

6 Как извлечь данные столбцов salary, l_name из таблицы employee_data?

Практическое занятие № 3 Работа с таблицами

Часть 1

Цель занятия: формирование навыка работы с таблицами, применение операторов сравнения, поиска текстовых данных по шаблону

Этапы выполнения работы:

1. Запустить MySql.
2. Выбрать БД employees.
3. Создать запросы, указанные ниже.
4. Продемонстрировать полученные запросы преподавателю, защитить выполненную работу.

ХОД РАБОТЫ

Теоретические сведения

Для выбора БД применяют оператор: use <имя_БД>.

Выборка данных с помощью условий

Полный формат оператора SELECT имеет вид:

```
SELECT имена_столбцов from имя_таблицы [WHERE ...условия];
```

В операторе SELECT условия являются необязательными.

Оператор сравнения на равенство (=) помогает выбрать одинаковые строки.

Поиск текстовых данных по шаблону осуществляется с помощью предложения `where` и оператора `LIKE`.Знак `%` действует как символ-заместитель. Он заменяет собой любую последовательность символов, например:

```
where title like '%про%';
```

Задания

1. Какой оператор используется для получения информации о таблице? Как используется этот оператор?
2. Как получить список всех баз данных, доступных в системе?
3. Напишите оператор SELECT для извлечения идентификационного номера сотрудников, которые старше 30 лет.
4. Напишите оператор SELECT для извлечения имен и фамилий всех Web-разработчиков.
5. Что выведет следующий оператор SELECT:

```
SELECT * from employee_data where salary <=100000;
```

6. Как вывести зарплаты и надбавки сотрудников, которые получают в качестве надбавок более 15000?

7. Перечислите имена всех сотрудников (фамилия, а затем имя), которые занимают должность бухгалтера.
8. Перечислить всех сотрудников, фамилии которых начинаются с буквы Р.
9. Вывести имена всех сотрудников в отделе продаж.
10. Что выведет следующий оператор:

```
SELECT f_name, l_name, salary from
employee_data where f_name like '%к%';
```

11. Перечислить фамилии и должности всех программистов.

Часть 2

Цель занятия: формирование навыка работы с таблицами, применение группировки и сортировки данных, удаление данных, шифрование пароля

Этапы выполнения работы:

1. Запустить MySql.
2. Выбрать БД employees.
3. Создать запросы, указанные ниже.
4. Продемонстрировать полученные запросы преподавателю, защитить выполненную работу.

ХОД РАБОТЫ

Теоретические сведения

1 Предложение HAVING

Чтобы вывести среднюю зарплату сотрудников в различных подразделениях (должностях), используется предложение GROUP BY, например:

```
select title, AVG(salary)
from employee_data
GROUP BY title;
```

2 Оператор удаления

Оператор удаления `DELETE` требует задания имени таблицы и необязательных условий.

```
DELETE from имя_таблицы [WHERE условия];
```

3 Оператор сортировки

Оператор сортировки данных `Order By` позволяет упорядочить данные по возрастанию (`Asc`) и по убыванию (`Desc`).

4 Шифрование пароля

Для шифрования пароля можно применить оператор `password`. Конструкция может иметь следующий вид:

```
Insert into users ('…', '…', password('….'));
```

```
Select * from users where name = 'Иван' and pass=password ('1234');
```

Задание

1. Вывести подразделения и средний возраст, где средний возраст больше 30.
2. Удалите Николая Серегина из таблицы.
3. Выведите данные о сотрудниках, отсортировав их по возрасту (по убыванию).
4. Выведите данные о сотрудниках, отсортировав их по зарплате (по возрастанию).
5. Что делает следующий оператор?

```
SELECT emp_id, l_name, title, age
from employee_data ORDER BY
title DESC, age ASC;
```

6. Создайте таблицу `users (логин, пароль).` Пароль зашифровать.

Практическое занятие № 4 Работа с таблицами

Цель занятия: формирование навыка работы с таблицами, применение различных операторов, создание нескольких таблиц, создание внешнего ключа.

Вид занятия: Самостоятельная работа

Этапы выполнения работы:
1. Запустить MySql.
2. Создать запросы, указанные ниже.
3. Продемонстрировать полученные запросы преподавателю, защитить выполненную работу.

ХОД РАБОТЫ

1. Создать базу данных Форум. Создать таблицу с 4 столбцами, в которых id_user - целочисленные значения, name - строковое значение varchar, email - строковое значение varchar, password - строковое значение varchar.

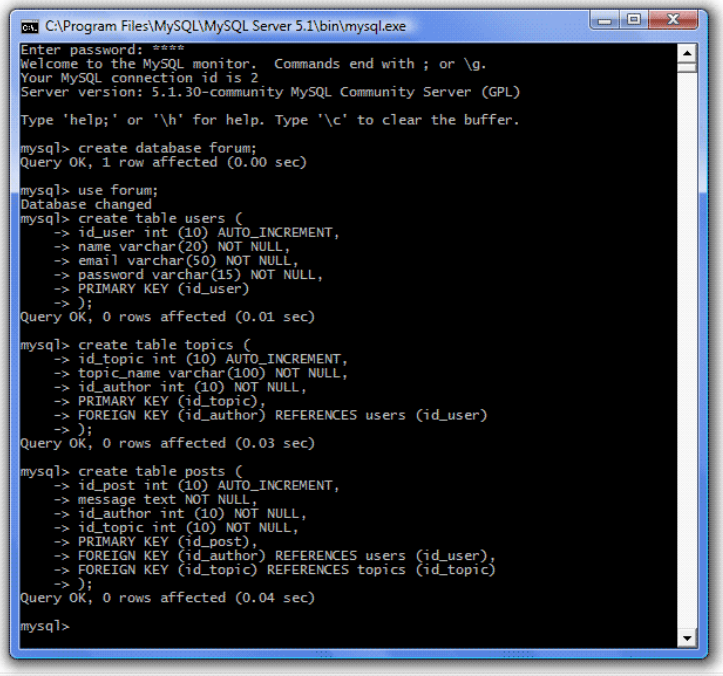
Все значения полей обязательны для заполнения, значит надо добавить тип NOT NULL	`id_user int (10) NOT NULL` `name varchar(20) NOT NULL` `email varchar(50) NOT NULL` `password varchar(15) NOT NULL`
Теперь надо указать, что поле id_user является первичным ключом. Для этого в SQL используется ключевое слово *PRIMARY KEY ()*, в скобочках указывается имя ключевого поля	`id_user int (10) AUTO_INCREMENT` `name varchar(20) NOT NULL` `email varchar(50) NOT NULL` `password varchar(15) NOT NULL`
Первый столбец является первичным ключом (т.е. его значения уникальны, и они однозначно идентифицируют запись). Следить за уникальностью самостоятельно можно, но не рационально.	`id_user int (10) AUTO_INCREMENT` `name varchar(20) NOT NULL` `email varchar(50) NOT NULL` `password varchar(15) NOT NULL` `PRIMARY KEY (id_user)`

Для этого в SQL есть специальный атрибут - *AUTO_INCREMENT*, который при обращении к таблице на добавление данных высчитывает максимальное значение этого столбца, полученное значение увеличивает на 1 и заносит его в столбец. Таким образом, в этом столбце автоматически генерируется уникальный номер, а следовательно тип NOT NULL излишен.	
Окончательный вариант таблицы:	`create table users (` `id_user int (10) AUTO_INCREMENT,` `name varchar(20) NOT NULL,` `email varchar(50) NOT NULL,` `password varchar(15) NOT NULL,` `PRIMARY KEY (id_user)` `);`

2. Создать таблицы `topics`, `posts`

Поле id_author является внешним ключом, т.е. оно может иметь только те значения, которые есть в поле id_user таблицы users. Для того, чтобы указать это в SQL есть ключевое слово *FOREIGN KEY ()*	`FOREIGN KEY (имя_столбца_которое_является_внешним_ключом) REFERENCES имя_таблицы_родителя (имя_столбца_родителя);`
Укажем, что id_author - внешний ключ:	`id_topic int (10) AUTO_INCREMENT` `topic_name varchar(100) NOT NULL` `id_author int (10) NOT NULL` `PRIMARY KEY (id_topic)` `FOREIGN KEY (id_author) REFERENCES users (id_user)`
Таблица `topics`:	`create table topics (` `id_topic int (10) AUTO_INCREMENT,` `topic_name varchar(100) NOT NULL,` `id_author int (10) NOT NULL,` `PRIMARY KEY (id_topic),` `FOREIGN KEY (id_author) REFERENCES users (id_user)` `);`
Таблица posts (сообщения). С двумя внешними ключами. Обратите внимание, внешних ключей у таблицы может быть несколько, а первичный ключ в MySQL может быть только один	`create table posts (` `id_post int (10) AUTO_INCREMENT,` `message text NOT NULL,` `id_author int (10) NOT NULL,` `id_topic int (10) NOT NULL,` `PRIMARY KEY (id_post),` `FOREIGN KEY (id_author) REFERENCES users (id_user),` `FOREIGN KEY (id_topic) REFERENCES topics (id_topic)` `);`

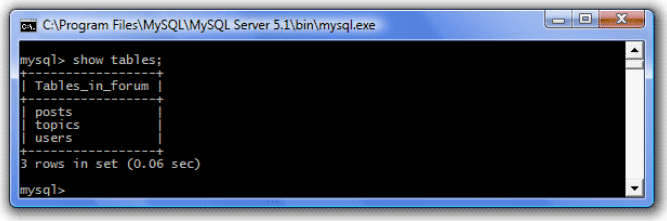
Запустите сервер MySQL и создайте три таблицы:



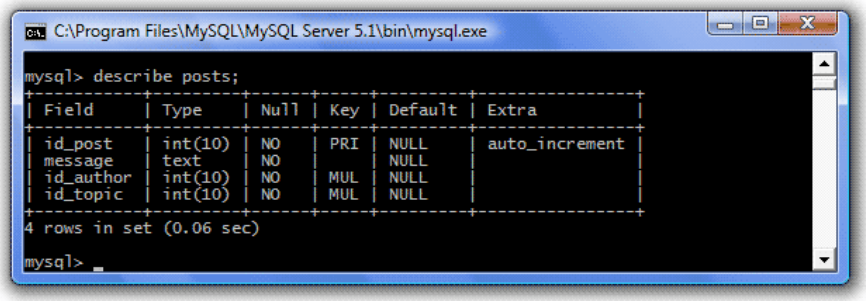
Обратите внимание, одну команду можно писать в несколько строк, используя клавишу Enter (MySQL автоматически подставляет символ новой строки ->), и только после разделителя (точки с запятой) нажатие клавиши Enter приводит к выполнению запроса.

Удалить таблицу или всю БД с помощью оператора DROP.

3. Отобразить таблицы - show tables:

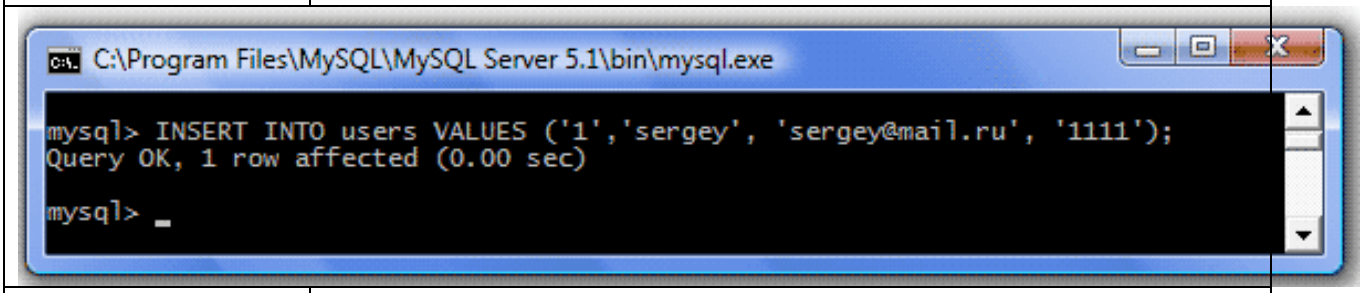


4. Посмотреть структуру последней таблицы posts:



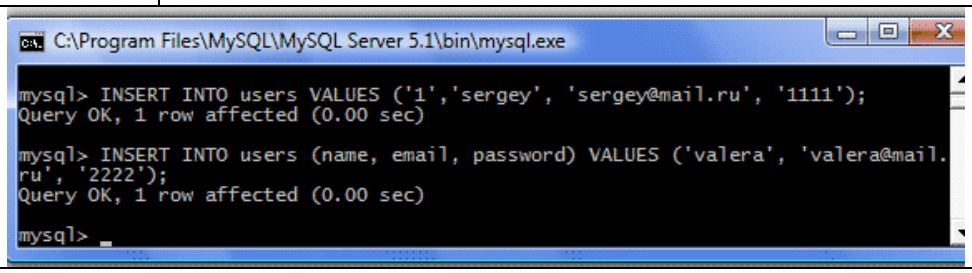
5. Внести данные в таблицы

Внесите таблицу users следующие значения	`INSERT INTO users VALUES ('1','sergey', 'sergey@mail.ru', '1111');`



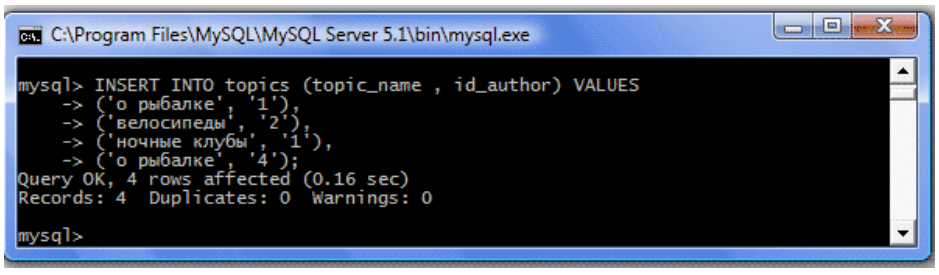
В таблице users все поля обязательны для заполнения, но первое поле имеет ключевое слово - AUTO_INCREMENT (т.е. оно заполняется автоматически), поэтому пропускаем этот столбец:

```
INSERT INTO users (name, email, password) VALUES ('valera', 'valera@mail.ru', '2222');
```

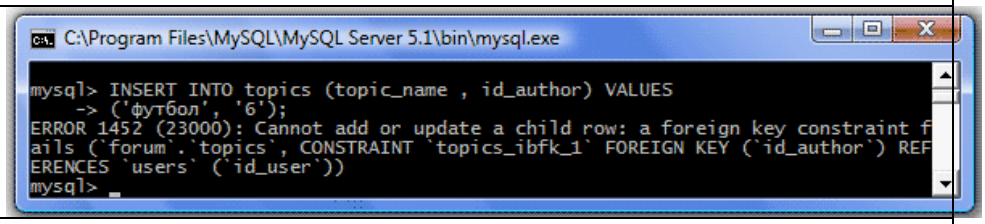


Если оставить пустым поле со значением NOT NULL, то сервер выдаст сообщение об ошибке и не выполнит запрос. Кроме того, при внесении данных сервер проверяет связи между таблицами. Поэтому вам не удастся внести в поле, являющееся внешним ключом, значение, отсутствующее в связанной таблице.

Внесите данные во вторую таблицу - topics (темы). Все тоже самое, но надо помнить, что значения в поле id_author должны присутствовать в таблице users (пользователи):



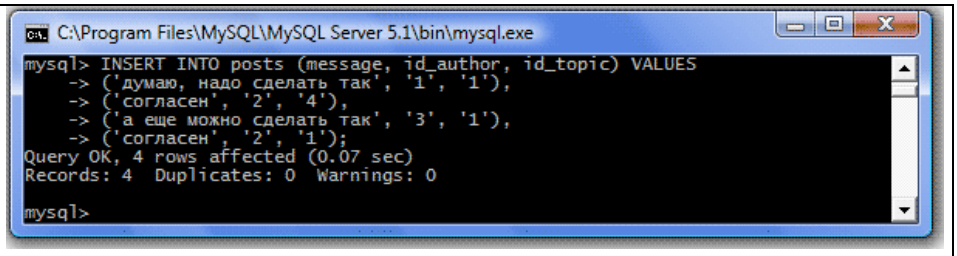
Внесите еще одну тему, но с id_author, которого в таблице users нет



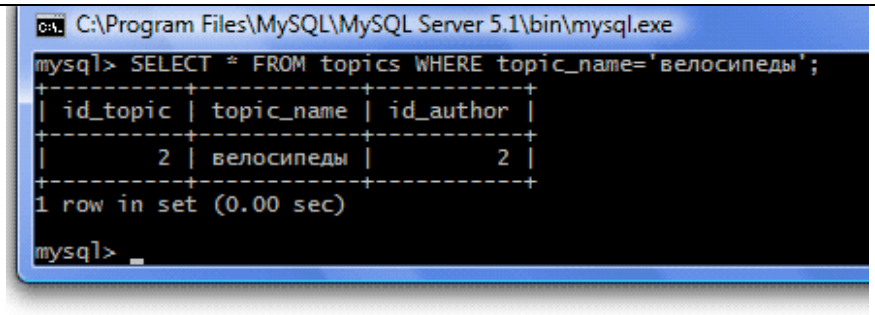
Сервер выдает ошибку и говорит, что не может внести такую строку, т.к. в поле, являющемся внешним ключом, стоит значение, отсутствующее в связанной таблице users.

Теперь внесем несколько строк в

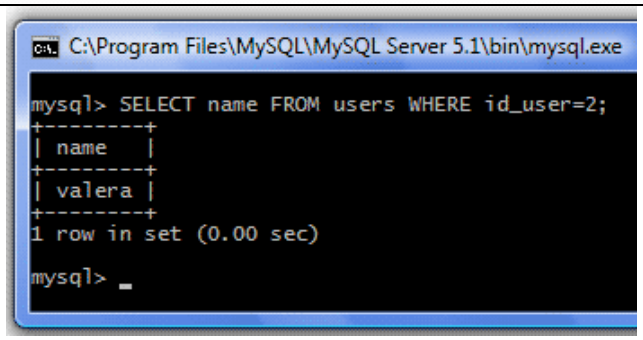
таблицу posts (сообщения), помня, что в ней у нас 2 внешних ключа, т.е. id_author и id_topic, которые мы будем вносить должны присутствовать в связанных с ними таблицах:



Узнать, кто создал тему "велосипеды"

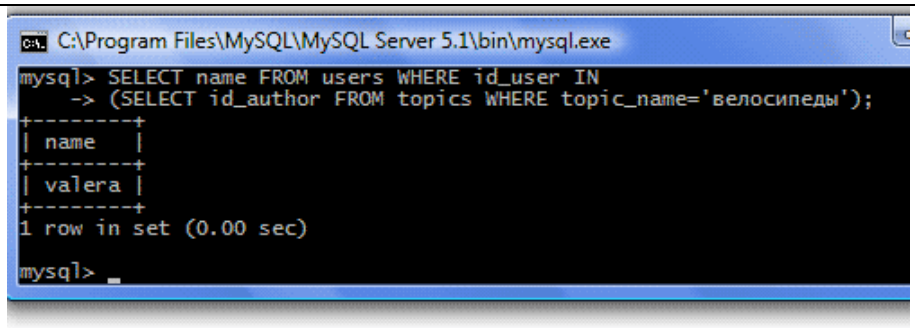


Вместо имени автора, мы получаем его идентификатор. Это и понятно, ведь мы делали запрос к одной таблице - Темы, а имена авторов тем хранятся в другой таблице - Пользователи. Поэтому, узнав идентификатор автора темы, нам надо сделать еще один запрос - к таблице Пользователи, чтобы узнать его имя:



В SQL предусмотрена возможность объединять такие запросы в один путем превращения одного из них в подзапрос (вложенный запрос).

Итак, чтобы узнать, кто создал тему "велосипеды", сделайте следующий запрос:



MySQL сначала обрабатывает подзапрос, возвращает id_author=2, и это значение передается в предложение WHERE внешнего запроса.

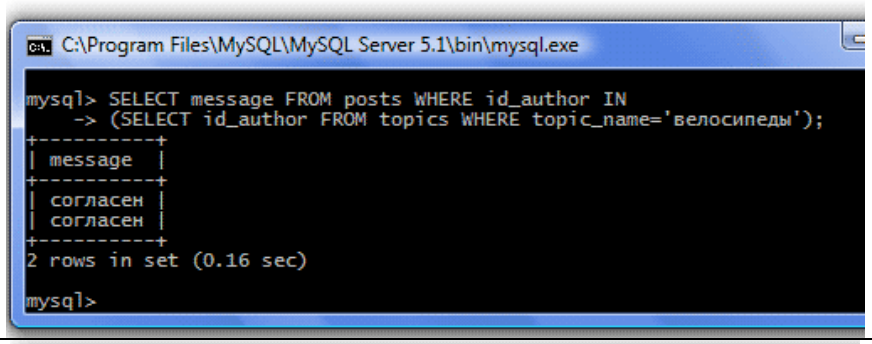
В одном запросе может быть несколько подзапросов, синтаксис у такого запроса следующий:

```
SELECT имя_столбца FROM имя_таблицы WHERE часть условия IN
  (SELECT имя_столбца FROM имя_таблицы WHERE часть условия IN
```

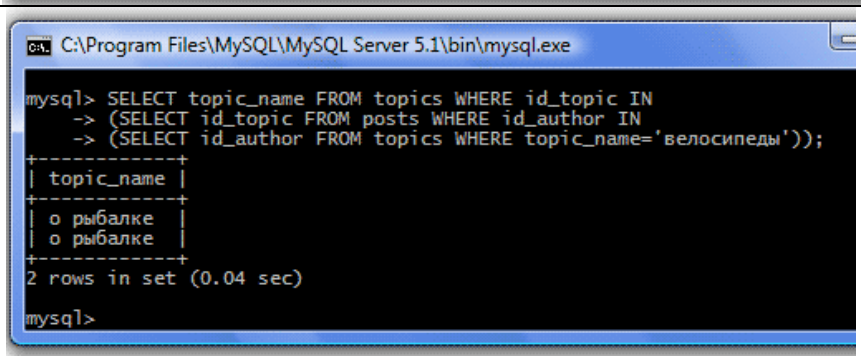
```
        (SELECT имя_столбца FROM имя_таблицы WHERE условие)
    )
;
```

Обратите внимание, что подзапросы могут выбирать только один столбец, значения которого они будут возвращать внешнему запросу. Попытка выбрать несколько столбцов приведет к ошибке.

Составьте запрос, который выдает, какие сообщения на форуме оставлял автор темы "велосипеды":



составьте запрос, который выдает, в каких темах оставлял сообщения автор темы "велосипеды":



- Сначала MySQL выполнит самый глубокий запрос:

```
SELECT id_author FROM topics WHERE topic_name='велосипеды'
```

- Полученный результат (id_author=2) передаст во внешний запрос, который примет вид:

```
SELECT id_topic FROM posts WHERE id_author IN (2);
```

- Полученный результат (id_topic:4,1) передаст во внешний запрос, который примет вид:

```
SELECT topic_name FROM topics WHERE id_topic IN (4,1);
```

- И выдаст окончательный результат (topic_name: о рыбалке, о рыбалке). Т.е. автор темы "велосипеды" оставлял сообщения в теме "О рыбалке", созданной Сергеем (id=1) и в теме "О рыбалке", созданной Светой (id=4).

Есть два момента, на которые стоит обратить внимание:

- Не рекомендуется создавать запросы со степенью вложения больше трех. Это приводит к увеличению времени выполнения и к сложности восприятия кода.
- Приведенный синтаксис вложенных запросов, скорее наиболее употребительный, но вовсе не единственный.

Например, вместо запроса

	`` SELECT name FROM users WHERE id_user IN (SELECT id_author FROM topics WHERE topic_name='велосипеды'); ``` можно использовать ``` SELECT name FROM users WHERE id_user = (SELECT id_author FROM topics WHERE topic_name='велосипеды'); ``` Т.е. можно использовать любые операторы, используемые с ключевым словом WHERE

САМОСТОЯТЕЛЬНАЯ РАБОТА

1. Вывести сообщение, которое создал 2 автор.
2. Вывести email и пароль для 1 и 3 пользователя.
3. Вывести автора, сообщение которого составляет меньше 7 символов.
4. Вывести тему, автор которой имеет email без цифр
5. Вывести автора темы «Шифрование»
6. Вывести сообщения, которые оставлял на форуме автор темы «Шифрование»
7. В каких темах оставлял сообщения автор темы «Шифрование»?

ВОПРОСЫ ДЛЯ КОНТРОЛЯ

1. Назначение команды Insert Into.
2. Синтаксис команды Insert.
3. Назначение команды Alter Table имя_таблицы Modify имя-поля новый_тип, Alter Table имя_таблицы Add имя-поля тип.
4. Назначение команды Select…From…
5. Назначение оператора Where.

Практическое занятие № 5

Часть 1 Логические операторы

Цель занятия: формирование навыка работы с таблицами, применение логических операторов

Этапы выполнения работы:

1. Запустить MySql.
2. Выбрать БД employees.
3. Создать запросы, указанные ниже.
4. Продемонстрировать полученные запросы преподавателю, защитить выполненную работу.

ХОД РАБОТЫ

Теоретические сведения

1 Булевы (логические) операторы:

1. AND – и;
2. OR – или;
3. NOT – не.

2 In и BETWEEN:

Чтобы найти сотрудников, которые являются разработчиками *Web* или системными администраторами, можно использовать оператор SELECT и логическую операцию OR. Но в *SQL* имеется более простой способ сделать это с помощью оператора IN (в множестве). Его использование не представляет никаких трудностей. Например:

```
SELECT f_name, l_name, title from
    -> employee_data where title
    -> IN ('разработчик Web', 'системный адм.');
```

Использование NOT перед IN позволяет вывести данные, которые не входят в множество, определяемое условием IN. Следующий оператор выводит *список* сотрудников, которые не занимают должность программиста или системного администратора.

```
SELECT f_name, l_name, title from
    -> employee_data where title NOT IN
    -> ('программист', 'системный адм.');
```

Оператор BETWEEN используется для определения целочисленных границ. Поэтому вместо age >= 32 AND age <= 40 можно использовать age BETWEEN 32 AND 40.

```
select f_name, l_name, age from
    -> employee_data where age BETWEEN
    -> 32 AND 40;
```

NOT также можно использовать вместе с BETWEEN, как в следующем операторе, который выводит сотрудников, зарплата которых меньше 90000 или больше 150000.

```
select f_name, l_name, salary
    -> from employee_data where salary
    -> NOT BETWEEN
    -> 90000 AND 150000;
```

3 Ограничение числа записей

По мере увеличения таблиц возникает необходимость вывода только подмножества данных. Этого можно добиться с помощью предложения LIMIT.

Например, чтобы вывести из таблицы имена только первых пяти сотрудников, используется оператор LIMIT с аргументом равным 5.

```
SELECT f_name, l_name from
employee_data LIMIT 5;
```

4 Извлечение подмножеств

LIMIT можно использовать также для извлечения подмножества данных, используя дополнительные аргументы.

Общая форма оператора LIMIT имеет следующий вид:

```
SELECT (что-нибудь) from таблица LIMIT начальная строка,
извлекаемое число записей;

SELECT f_name, l_name from
employee_data LIMIT 6,3;
```

5 Исключение появления повторяющихся данных

```
select title from employee_data;
```

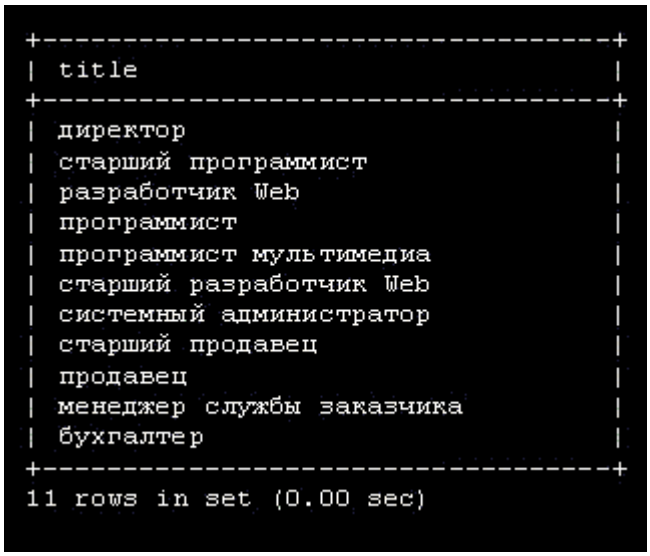
На рисунке приведен результат запроса.

```
+----------------------------------+
| title                            |
+----------------------------------+
| директор                         |
| старший программист              |
| старший программист              |
| разработчик Web                  |
| разработчик Web                  |
| программист                      |
| программист                      |
| программист                      |
| программист                      |
| программист мультимедиа          |
| программист мультимедиа          |
| программист мультимедиа          |
| старший разработчик Web          |
| системный администратор          |
| системный администратор          |
| старший продавец                 |
| продавец                         |
| продавец                         |
| продавец                         |
| менеджер службы заказчика        |
| бухгалтер                        |
+----------------------------------+
21 rows in set (0.00 sec)
```

Можно видеть, что *список* содержит повторяющиеся данные. Предложение *SQL* DISTINCT выводит только уникальные данные. Вот как оно используется.

```
select DISTINCT title from employee_data;
```

На рисунке приведен результат запроса. Все должности базы данных компании без повторов.



```
+-----------------------------------+
| title                             |
+-----------------------------------+
| директор                          |
| старший программист               |
| разработчик Web                   |
| программист                       |
| программист мультимедиа           |
| старший разработчик Web           |
| системный администратор           |
| старший продавец                  |
| продавец                          |
| менеджер службы заказчика         |
| бухгалтер                         |
+-----------------------------------+
11 rows in set (0.00 sec)
```

6 Изменение записей

Команда `UPDATE` выполняет изменение данных в таблицах. Она имеет очень простой формат.

```
UPDATE имя_таблицы SET
имя_столбца_1 = значение_1,
имя_столбца_2 = значение_2,
имя_столбца_3 = значение_3, ...
[WHERE условия];
```

Задания

1. Вывести имена и фамилии всех сотрудников, которые получают зарплату не более 90000 и не являются программистами, старшими программистами или программистами мультимедиа.
2. Что делает следующий оператор?

```
SELECT l_name, f_name from employee_data
where title NOT LIKE '%продавец%' AND age < 30;
```

3. Вывести все идентификационные номера и имена сотрудников в возрасте между 32 и 40 годами.
4. Выберите имена всех сотрудников в возрасте 32 лет, которые не являются программистами.
5. Найдите всех сотрудников, которые занимают должность "старший программист" и "программист мультимедиа".
6. Выведите список имен сотрудников, зарплата которых составляет от 70000 до 90000.
7. Что делает следующий оператор?

```
SELECT f_name, l_name, title from
employee_data where title NOT IN ('программист', 'старший
программист','программист мультимедиа');
```

8. Вот более сложный оператор, который объединяет `BETWEEN` и `IN`. Что он делает?
9. Вывести список сотрудников (фамилию и имя), которые занимают должность "программист" или "разработчик Web" и отсортировать их фамилии по алфавиту.
10. Найдите имена 5 самых молодых сотрудников компании.
11. Извлеките 5 записей, начиная с 10 строки.
12. Выведите имя, фамилию и зарплату сотрудника, который получает самую большую зарплату.

13. Что делает следующий оператор?

```
SELECT emp_id, age, perks
from employee_data ORDER BY
perks DESC LIMIT 10;
```

14. Сколько уникальных вариантов зарплаты имеется в компании? Представьте их в убывающем порядке.
15. Сколько различных имен имеется в базе данных?
16. Измените фамилию Чащина на Петрова. Внесите соответствующие изменения в базу данных.
17. Название должности "программист мультимедиа" необходимо изменить на "специалист по мультимедиа".
18. Увеличьте зарплату всем сотрудниками (кроме директора) на 10000.

Практическое занятие № 5

Часть 2 Команды обработки данных

Цель занятия: формирование навыка работы с таблицами, применяя команды обработки данных: поиск минимального и максимального значений; поиск среднего значения и суммы; вычисление среднего значения; именование столбцов; подсчет числа записей.

Этапы выполнения работы:

1. Запустить MySql.
2. Выбрать БД employees.
3. Создать запросы, указанные ниже.
4. Продемонстрировать полученные запросы преподавателю, защитить выполненную работу.

ХОД РАБОТЫ

В *MySQL* имеются встроенные функции для вычисления минимального и максимального значений.

SQL имеет 5 агрегатных функций.

1. `MIN()`: минимальное значение
2. `MAX()`: максимальное значение
3. `SUM()`: сумма значений
4. `AVG()`: среднее значений
5. `COUNT()`: подсчитывает число записей

В этом параграфе мы рассмотрим *поиск* минимального и максимального значений столбца.

1 Минимальное значение

```
select MIN(salary) from employee_data;
```

На рис. приведен результат запроса.

```
+---------------+
| MIN(salary)   |
+---------------+
|     70000     |
+---------------+
1 row in set (0.00 sec)
```

Рис. 1. Поиск минимальной зарплаты

2 Максимальное значение

```
select MAX(salary) from employee_data;
```

На рис. 2. приведен результат запроса.

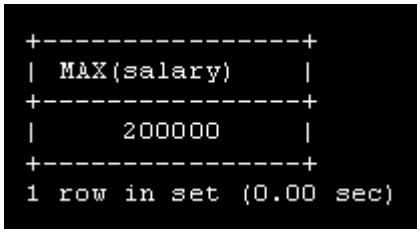


Рис. 2. Поиск максимальной зарплаты

3 Поиск среднего значения и суммы

Суммирование значений столбца с помощью функции SUM

Агрегатная функция `SUM()` *вычисляет общую сумму* значений в столбце. Для этого необходимо задать имя столбца, которое должно быть помещено внутри скобок.

Давайте посмотрим, сколько компания BigFoot тратит на зарплату своих сотрудников.

```
select SUM(salary) from employee_data;
```

На рис. 3 приведен результат запроса.

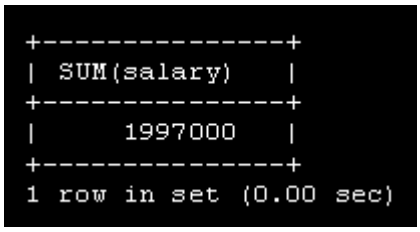


Рис. 3 Сумма всех зарплат

Аналогично можно вывести общую сумму надбавок, выдаваемых сотрудникам.

```
select SUM(perks) from employee_data;
```

На рис. 4 приведен результат запроса.

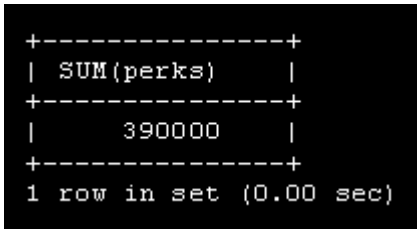


Рис. 4 Сумма всех надбавок

Можно найти также общую сумму зарплаты и надбавок.

```
select sum(salary) + sum(perks) from employee_data;
```

На рис. 5 приведен результат запроса.

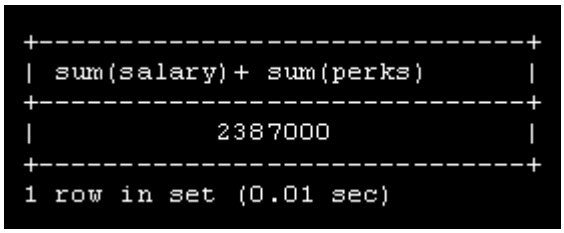


Рис. 5 Общая сумма зарплаты и надбавок

Здесь показаны также дополнительные возможности команды SELECT. Значения можно складывать, вычитать, умножать или делить. В действительности можно записывать полноценные арифметические выражения.

4 Вычисление среднего значения

Агрегатная функция `AVG()` используется для вычисления среднего значения данных в столбце.

```
select avg(age) from employee_data;
```

На рис. 6 приведен результат запроса.

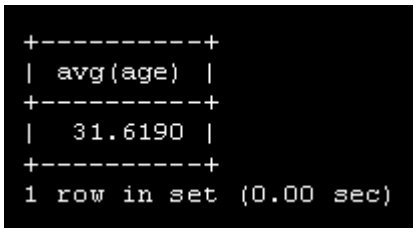


Рис. 6 Средний возраст сотрудников

Пример выше вычисляет средний возраст сотрудников компании BigFoot, а следующий выводит среднюю зарплату.

```
select avg(salary) from employee_data;
```

На рис. 7 приведен результат запроса.

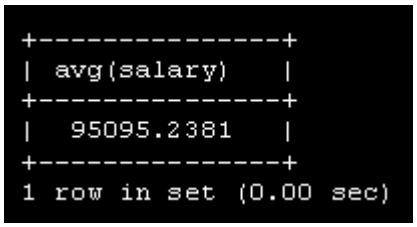


Рис. 7 Средняя зарплата сотрудников

MySQL позволяет задавать имена для выводимых столбцов. Поэтому вместо `f_name` или `l_name` и т.д. можно использовать более понятные и наглядные термины. Это делается с помощью оператора `AS`.

```
select avg(salary) AS
'Средняя зарплата' from
employee_data;
```

На рис. 8 приведен результат запроса.

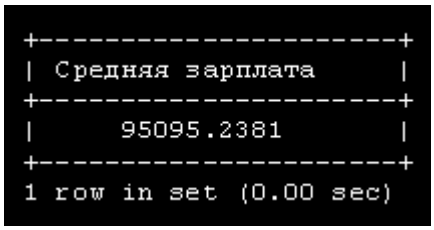


Рис. 8 Вывод средней зарплаты с использованием псевдо-имен столбцов.

Такие псевдо-имена могут сделать *вывод* более понятным для пользователей. Важно только помнить, что при задании псевдо-имен с пробелами необходимо заключать такие имена в кавычки. Вот еще один пример:

```
select (SUM(perks)/SUM(salary) * 100)
AS 'Процент надбавок' from
employee_data;
```

На рис. 9 приведен результат запроса.

```
+----------------------+
| Процент надбавок     |
+----------------------+
|         19.53        |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 9 Вывод процента зарплаты, которую сотрудники получают в качестве надбавок с использованием псевдо-имен

5 Подсчет числа записей

Агрегатная *функция* `COUNT()` подсчитывает и выводит общее число записей. Например, чтобы подсчитать общее число записей в таблице, выполните следующую команду.

```
select COUNT(*) from employee_data;
```

На рис. 10 приведен результат запроса.

```
+--------------+
| COUNT(*)     |
+--------------+
|      21      |
+--------------+
1 row in set (0.00 sec)
```

Рис. 10 Общее количество записей

Как мы уже знаем, знак * означает "все данные".

Теперь давайте подсчитаем общее число сотрудников, которые занимают должность "программист".

```
select COUNT(*) from employee_data
where title = 'программист';
```

На рис. 11 приведен результат запроса.

```
+--------------+
| COUNT(*)     |
+--------------+
|       4      |
+--------------+
1 row in set (0.01 sec)
```

Рис 11 Общее количество сотрудников-программистов

6 Группировка данных

Предложение `GROUP BY` позволяет группировать аналогичные данные. Поэтому, чтобы вывести все уникальные должности в таблице, можно выполнить команду

```
select title from employee_data
GROUP BY title;
```

На рис. 12 приведен результат запроса.

```
+----------------------------------------+
| title                                  |
+----------------------------------------+
| директор                               |
| менеджер по работе с заказчиком        |
| бухгалтер                              |
| продавец                               |
| программист мультимедиа                |
| программист                            |
| старший продавец                       |
| старший программист                    |
| старший разработчик Web                |
| системный администратор                |
| разработчик Web                        |
+----------------------------------------+
11 rows in set (0.01 sec)
```

Рис. 12 Все уникальные должности сотрудников

Вот как можно подсчитать число сотрудников имеющих определенную должность.

```
select title, count(*)
from employee_data GROUP BY title;
```

На рис. 13 приведен результат запроса.

```
+----------------------------------------+----------+
| title                                  | count(*) |
+----------------------------------------+----------+
| директор                               | 1        |
| менеджер по работе с заказчиком        | 1        |
| бухгалтер                              | 1        |
| продавец                               | 3        |
| программист мультимедиа                | 3        |
| программист                            | 4        |
| старший продавец                       | 1        |
| старший программист                    | 2        |
| старший разработчик Web                | 1        |
| системный администратор                | 2        |
| разработчик Web                        | 2        |
+----------------------------------------+----------+
11 rows in set (0.00 sec)
```

Рис. 13 Количество сотрудников по должностям

В предыдущей команде *MySQL* сначала создает группы различных должностей, а затем выполняет подсчет в каждой группе.

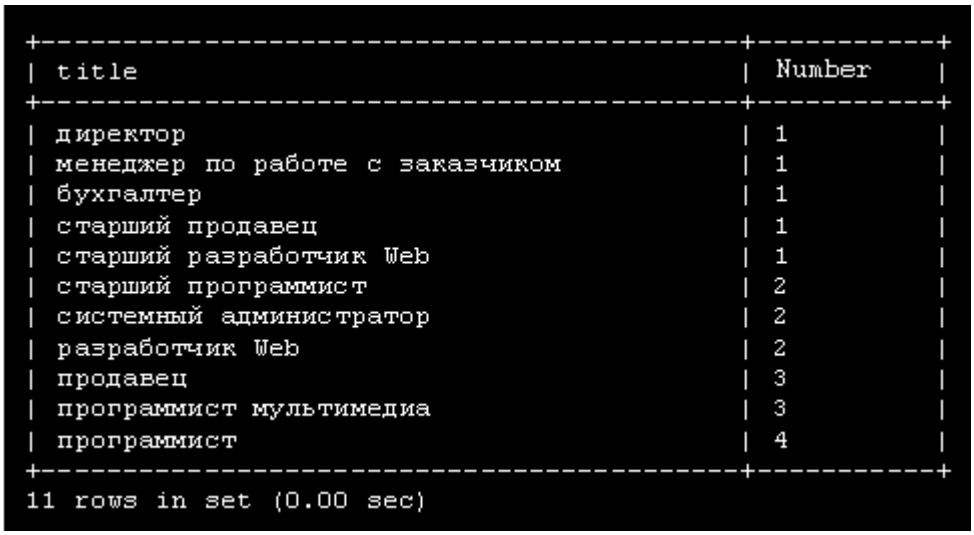
7 Сортировка данных

Теперь давайте найдем и выведем число сотрудников, имеющих различные должности, и отсортируем их с помощью `ORDER BY`.

```
select title, count(*) AS Number
```

```
from employee_data
GROUP BY title
ORDER BY Number;
```

На рис. 14 приведен результат запроса.



```
+-----------------------------------------+-----------+
| title                                   | Number    |
+-----------------------------------------+-----------+
| директор                                | 1         |
| менеджер по работе с заказчиком         | 1         |
| бухгалтер                               | 1         |
| старший продавец                        | 1         |
| старший разработчик Web                 | 1         |
| старший программист                     | 2         |
| системный администратор                 | 2         |
| разработчик Web                         | 2         |
| продавец                                | 3         |
| программист мультимедиа                 | 3         |
| программист                             | 4         |
+-----------------------------------------+-----------+
11 rows in set (0.00 sec)
```

Рис. 14 Количество сотрудников по должностям с сортировкой

Задания

1. Найдите минимальные надбавки.
2. Найдите максимальную зарплату среди всех "программистов".
3. Найдите возраст самого старого "продавца".
4. Найдите имя и фамилию самого старого сотрудника.
5. Вывести сумму всех возрастов сотрудников, работающих в компании BigFoot.
6. Как вычислить общее количество лет стажа работы сотрудников в компании BigFoot?
7. Вычислите сумму зарплат и средний возраст сотрудников, которые занимают должность "программист".
8. Что делает следующий оператор?

```
select (SUM(perks)/SUM(salary) * 100)
from employee_data;
```

9. Подсчитайте число сотрудников, которые проработали в BigFoot более трех лет.
10. Подсчитайте количество сотрудников в группах одного возраста.
11. Измените предыдущее задание так, чтобы возраст выводился в убывающем порядке.
12. Найдите средний возраст сотрудников в различных подразделениях (должностях).
13. Измените предыдущий оператор так, чтобы данные выводились в убывающем порядке среднего возраста.

Практическое занятие № 6

Часть 1 Математические функции

Цель занятия: формирование навыка работы с запросами, применяя математические функции

Этапы выполнения работы:

1. Запустить MySql.
2. Выбрать БД employees.
3. Создать запросы, указанные ниже.
4. Продемонстрировать полученные запросы преподавателю, защитить выполненную работу.

ХОД РАБОТЫ

Описанные ниже функции выполняют различные математические *операции*. В качестве аргументов большинство из них принимает числа с плавающей запятой и возвращает результат аналогичного типа.

ABS(число)

Эта *функция* возвращает *модуль* числа

На рис. 1а и 1б приведены примеры работы с функцией ABS.

```
mysql> SELECT ABS(-4.05022);

+----------------------+
| ABS(-4.05022)        |
+----------------------+
|        4.05022       |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 1(а). Модуль числа

```
mysql> SELECT ABS(2);

+----------------------+
| ABS (2)              |
+----------------------+
|            2         |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 1(б). Модуль числа

ASIN(число)

Эта *функция* возвращает арксинус числа. *Диапазон* допустимых значений – от -1 до 1. Вне этого диапазона *значение* арксинуса не определено.

На рис. 2а, 2б и 2в приведены примеры работы с функцией ASIN.

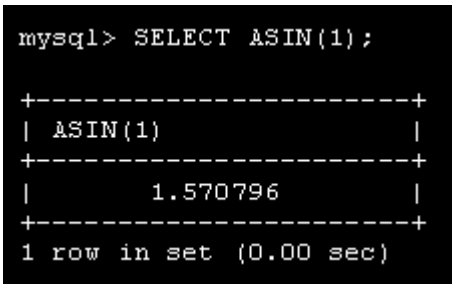


Рис. 2(а). Арксинус числа

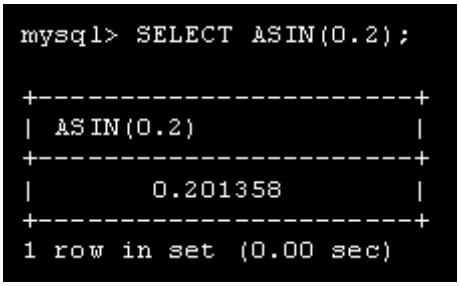


Рис. 2(б). Арксинус числа

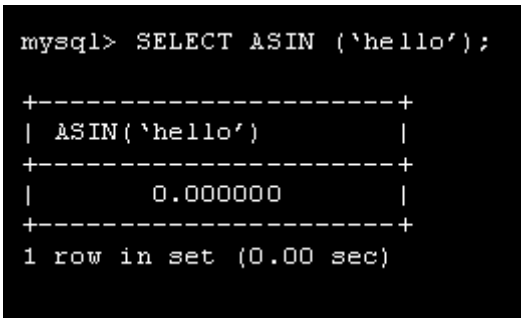


Рис. 2(в). Арксинус числа

ACOS(число)

Эта *функция* возвращает арккосинус числа.

Диапазон допустимых значений – от -1 до 1. Вне этого диапазона *значение* арккосинуса не определено.

На рис. 3а, 3б и 3в приведены примеры работы с функцией ACOS.

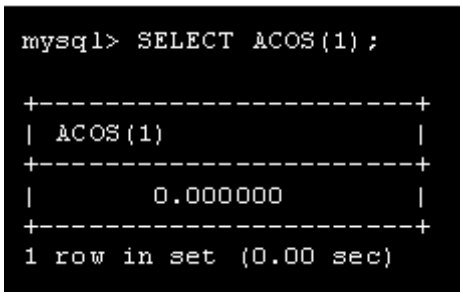


Рис. 3(а). Арккосинус числа

```
mysql> SELECT ACOS(1.0001);

+----------------------+
| ACOS(1.0001)         |
+----------------------+
|         NULL         |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 3(б). Арккосинус числа

```
mysql> SELECT ACOS(0);

+----------------------+
| ACOS(0)              |
+----------------------+
|       1.570796       |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 3(в). Арккосинус числа

ATAN(число)

Эта *функция* возвращает арктангенс числа.

На рис. 4а, 4(б) и 4(в) приведены примеры работы с функцией ATAN.

```
mysql> SELECT ATAN(1);

+----------------------+
| ATAN(1)              |
+----------------------+
|       0.785398       |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 4(а). Арктангенс числа

```
mysql> SELECT ATAN(2);

+----------------------+
| ATAN(2)              |
+----------------------+
|       1. 107149      |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 4(б). Арктангенс числа

```
mysql> SELECT ATAN(-2);

+----------------------+
| ATAN(-2)             |
+----------------------+
|       -1. 107149     |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 4(в). Арктангенс числа

ATAN2(число1, число2)

Эта *функция* возвращает угол в радианах точки с заданными координатами.

На рис. 5(а), 5(б) и 5(в) приведены примеры работы с функцией ATAN2.

```
mysql> SELECT ATAN2(3, 7);

+----------------------+
| ATAN2 (3, 7)         |
+----------------------+
|       0.404892       |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 5(а). Угол по координатам точки

```
mysql> SELECT ATAN2(-2, 2);

+----------------------+
| ATAN2 (-2, 2)        |
+----------------------+
|       -0.785398      |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 5(б). Угол по координатам точки

```
mysql> SELECT ATAN2(PI(), 0);

+----------------------+
| ATAN2(PI(), 0)       |
+----------------------+
|       1.570796       |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 5(в). Угол по координатам точки

CEILING(число)

CEIL(число)

Эта *функция* округляет число до ближайшего большего целого числа.

На рис. 6(а), 6(б) и 6(в) приведены примеры работы с функцией CEIL.

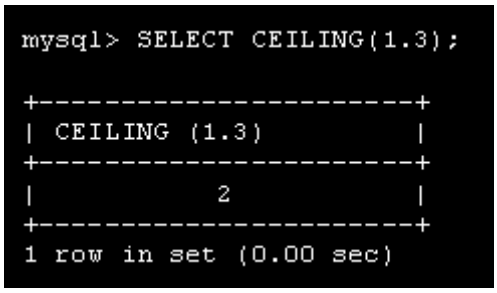


Рис. 6(а). Функция CEIL

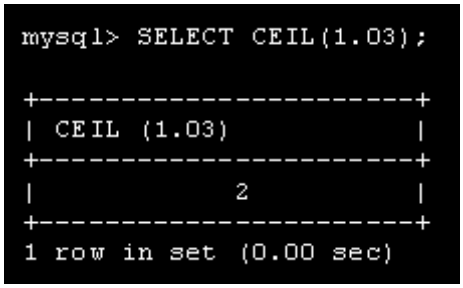


Рис. 6(б). Функция CEIL

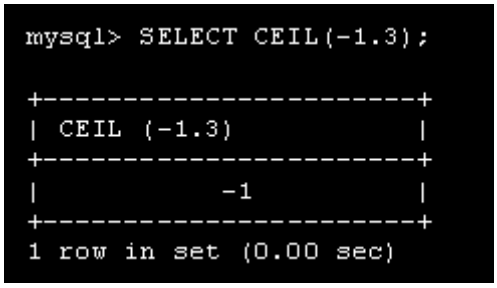


Рис. 6(в). Функция CEIL

COS(число)

Возвращает косинус числа

На рис. 7 приведен пример работы с функцией COS.

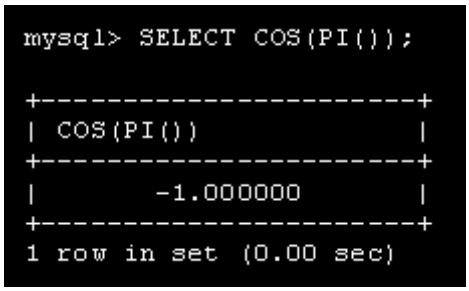


Рис. 7. Косинус числа

COT(число)

Возвращает котангенс числа.

На рис. 8(а) и 8(б) приведены примеры работы с функцией *COT*.

```
mysql> SELECT COT(12);

+----------------------+
| COT(12)              |
+----------------------+
|      -1.572 67341    |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 8(а). Котангенс числа

```
mysql> SELECT COT(0);
+----------------------+
| COT(0)               |
+----------------------+
|          NULL        |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 8(б). Котангенс числа

CRC32(выражение)

Вычисляет проверочное *значение* в циклическом избыточном коде и возвращает 32-разрядное целое. Результат равен NULL, если передается *аргумент* NULL. Ожидается, что *аргумент* будет строкой, и будет рассматриваться в качестве таковой в противном случае.

На рис. 9 приведен пример работы с функцией CRC32.

```
mysql> SELECT CRC32('MySQL');
+----------------------+
| CRC32('MySQL')       |
+----------------------+
|       3259397556     |
+----------------------+
1 row in set (0.00 sec)
```

Рис..9. Циклический избыточный код

DEGREES(число)

Возвращает *аргумент*, преобразованный из радианов в градусы.

На рис. 10 приведен пример работы с функцией DEGREES.

```
mysql> SELECT DEGREES(PI());
+----------------------+
| DEGREES(PI())        |
+----------------------+
|       180.000000     |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 10. Преобразование из радианов в градусы

EXP(число)

Эта *функция* возводит число **e** (*основание* натурального логарифма) в заданную степень.

На рис. 11(а) и 11(б) приведены примеры работы с функцией EXP.

```
mysql> SELECT EXP(2);

+------------------+
|      EXP(2)      |
+------------------+
|     7.389056     |
+------------------+
1 row in set (0.00 sec)
```

Рис. 11(а). Экспонента

```
mysql> SELECT EXP(-2);

+-------------------+
|      EXP(-2)      |
+-------------------+
|     0.135335      |
+-------------------+
1 row in set (0.00 sec)
```

Рис. 11(б). Экспонента

FLOOR(число)

Эта *функция* округляет число до ближайшего меньшего целого числа.

На рис. 12(а), 12(б) и 12(в) приведены примеры работы с функцией FLOOR.

```
mysql> SELECT FLOOR(1.7);

+-------------------+
|   FLOOR (1.7)     |
+-------------------+
|         1         |
+-------------------+
1 row in set (0.00 sec)
```

Рис. 12(а). Функция FLOOR

```
mysql> SELECT FLOOR(1.23);

+--------------------+
|    FLOOR (1.23)    |
+--------------------+
|         1          |
+--------------------+
1 row in set (0.00 sec)
```

Рис. 12(б). Функция FLOOR

```
mysql> SELECT FLOOR(-1.23);

+-----------------------+
|      FLOOR (-1.23)    |
+-----------------------+
|          -2           |
+-----------------------+
1 row in set (0.00 sec)
```

Рис. 12(в). Функция FLOOR

GREATEST(...)

Эта *функция* возвращает наибольшее *значение* из списка. Она может работать как с числами, так и со строками.

На рис. 13 приведен пример работы с функцией GREATEST.

```
mysql> SELECT GREATEST (1, 2, 3);

+-----------------------+
|   GREATEST (1, 2, 3)  |
+-----------------------+
|           3           |
+-----------------------+
1 row in set (0.00 sec)
```

Рис. 13. Наибольшее значение из списка

LEAST(...)

Функция возвращает наименьшее *значение* из списка.

На рис. 14 приведен пример работы с функцией LEAST.

```
mysql>  SELECT LEAST (1, 2, 3);

+-----------------------+
|    LEAST (1, 2, 3)    |
+-----------------------+
|           1           |
+-----------------------+
1 row in set (0.00 sec)
```

Рис. 14. Наименьшее значение из списка

LN(число)

LOG(число)

Эта *функция* возвращает натуральный логарифм числа.

На рис. 15(а) и 15(б) приведены примеры работы с функцией LN.

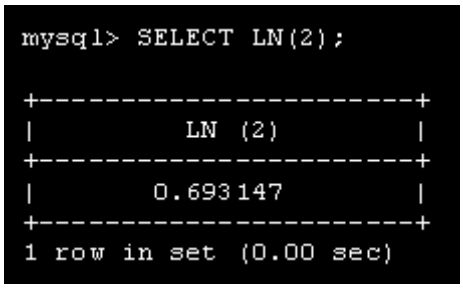


Рис. 15(а). Натуральный логарифм числа

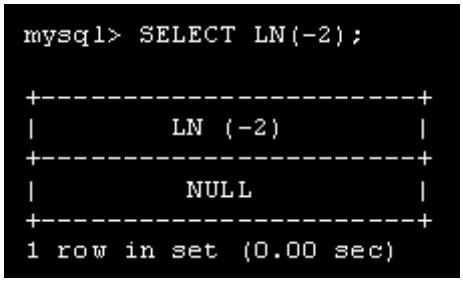


Рис. 15(б). Натуральный логарифм числа

LOG(число1, число2)

При вызове с одним параметром *функция* LOG возвращает натуральный логарифм числа, а при вызове с двумя параметрами - возвращает логарифм числа2 *по* основанию число1.

На рис. 16(а) и 16(б) приведены примеры работы с функцией LOG2.

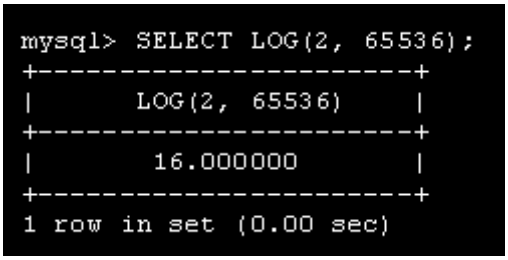


Рис. 16(а). Логарифм числа по основанию

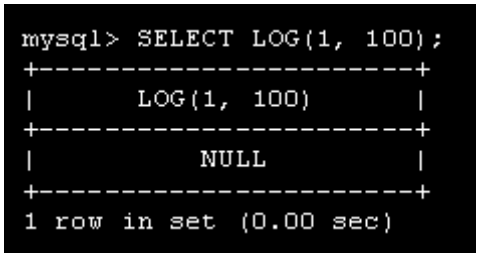


Рис. 16(б). Логарифм числа по основанию

LOG(число1, число2) эквивалентна LOG(число2) / LOG(число1).

LOG2(число)

Возвращает логарифм числа *по* основанию 2.

На рис. 17(а) и 17(б) приведены примеры работы с функцией LOG.

```
mysql> SELECT LOG2(65536);
+----------------------+
|      LOG2(65536)     |
+----------------------+
|         16.000000    |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 17(а). Логарифм числа по основанию 2

```
mysql> SELECT LOG2(-100);
+----------------------+
|      LOG2(-100)      |
+----------------------+
|          NULL        |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 17(б). Логарифм числа по основанию 2

Функция LOG2() удобна для того, чтобы определить, сколько *бит* потребуется для сохранения числа. Вместо нее можно использовать LOG(число) /LOG(2).

LOG10(число)

Возвращает логарифм числа *по* основанию 10.

На рис. 18(а), 18(б) и 18(в) приведены примеры работы с функцией LOG10.

```
mysql> SELECT LOG10(2);
+----------------------+
|        LOG10(2)      |
+----------------------+
|        0.301030      |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 18(а). Десятичный логарифм

```
mysql> SELECT LOG10(100);
+----------------------+
|      LOG10(100)      |
+----------------------+
|       2.000000       |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 18(б). Десятичный логарифм

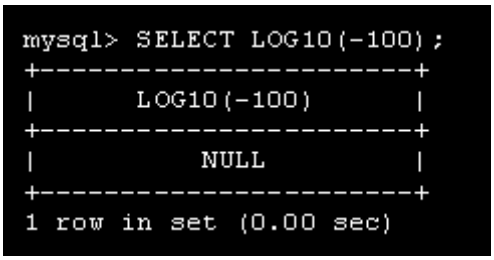


Рис. 18(в). Десятичный логарифм

MOD(число1, число2)

число1 % число2

число1 MOD число2

Эта *функция* возвращает *остаток от деления* первого числа на второе подобно оператору %.

На рис. 19(а), 19(б), 19(в) и 19(г) приведены примеры работы с функцией MOD.

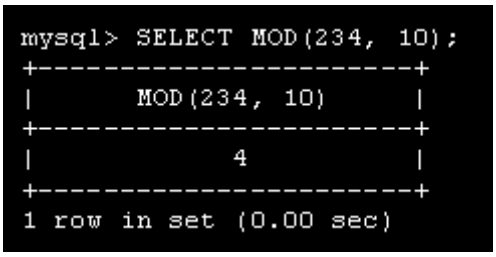


Рис. 19(а). Остаток от деления

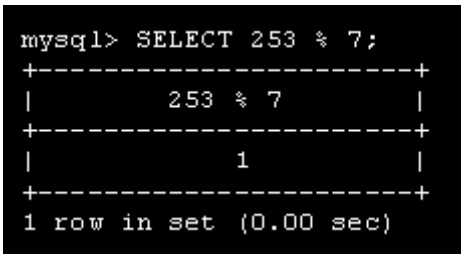


Рис. 19(б). Остаток от деления

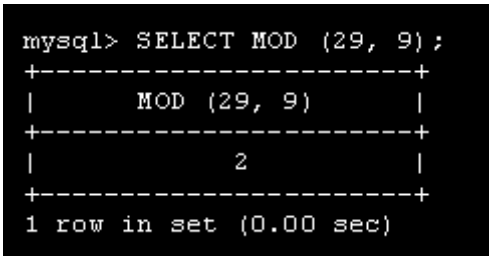


Рис. 19(в). Остаток от деления

```
mysql> SELECT 29 MOD 9;
+----------------------+
|        29 MOD 9      |
+----------------------+
|           2          |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 19(г). Остаток от деления

PI()

Возвращает *значение* числа π . *По* умолчанию отображается пять знаков после десятичной запятой, но внутренне MySQL использует полное *представление* действительного числа двойной точности.

На рис. 20(а) и 20(б) приведены примеры работы с функцией PI.

```
mysql> SELECT PI();
+----------------------+
|          PI()        |
+----------------------+
|        3.141593      |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 20(а). Число Пи

```
mysql> SELECT PI ()+0.000000000000000000;
+-----------------------------------------+
|        PI ()+0.000000000000000000       |
+-----------------------------------------+
|              3.141592653589793116       |
+-----------------------------------------+
1 row in set (0.00 sec)
```

Рис. 20(б). Число Пи

POW(число1, число2)

POWER(число1, число2)

Возвращает *значение* число1, возведенное в степень число2.

На рис. 21(а), 21(б) и 21(в) приведены примеры работы с функцией POW.

```
mysql> SELECT POW(2,2);
+----------------------+
|        POW(2,2)      |
+----------------------+
|        4.000000      |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 21(а). Возведение числа в степень

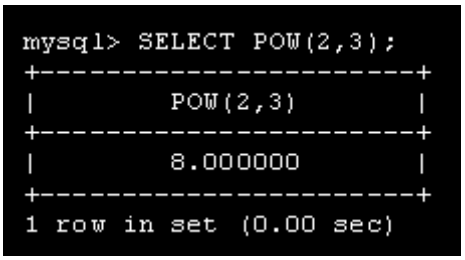


Рис.21(б). Возведение числа в степень

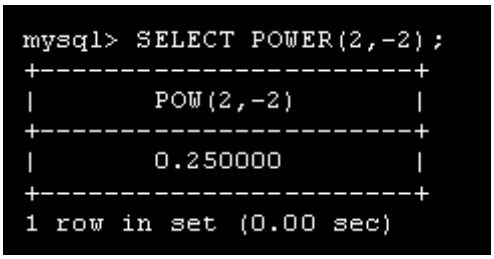


Рис. 21(в). Возведение числа в степень

RADIANS(число)

Возвращает *аргумент*, преобразованный из градусов в радианы.

На рис.22(а) и 22(б) приведены примеры работы с функцией RADIANS.

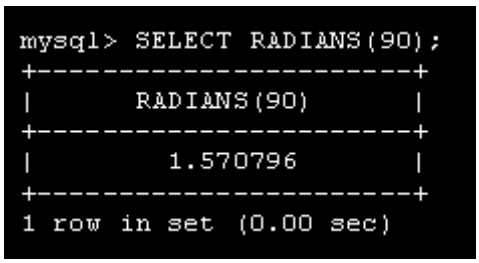


Рис. 22(а). Преобразование из градусов в радианы

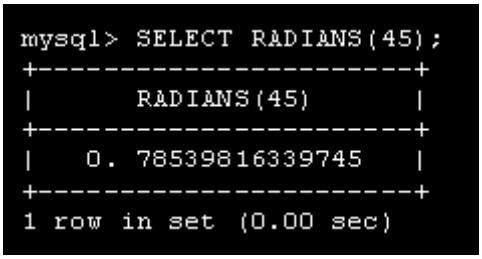


Рис. 22(б). Преобразование из градусов в радианы

RAND([число])

Возвращает случайное число двойной точности в диапазоне от 0 до 1. Если указан целочисленный *аргумент*, он служит начальным числом для генератора случайных чисел (генерируя повторяющуюся последовательность). Если *аргумент* отсутствует, используется *значение* системных часов.

На рис. 23(а) и 23(б) приведены примеры работы с функцией RAND.

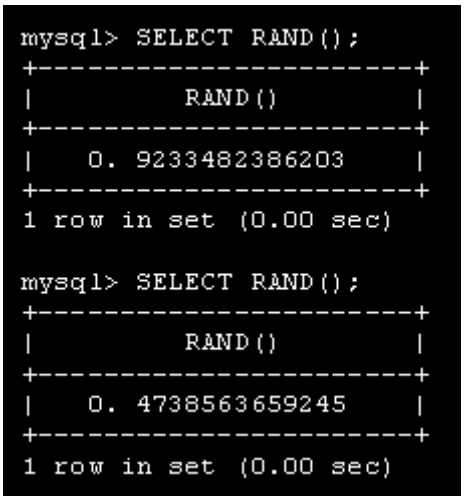


Рис. 23(а). Создание случайных чисел

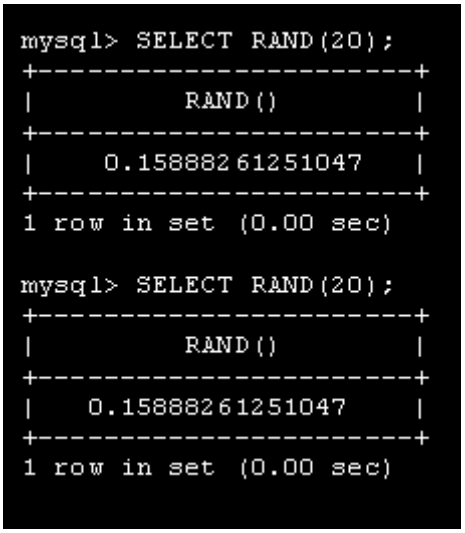


Рис. 23(б). Создание случайных чисел

Функцию можно использовать для извлечения строк в случайном порядке.

mysql> SELECT * FROM имя_таблицы ORDER BY RAND();

ORDER BY RAND() в комбинации с LIMIT удобно для выбора случайного примера из набора строк:

mysql> SELECT * FROM tablel, table2 WHERE a=b AND c<d
-> ORDER BY RAND() LIMIT 1000;

Следует отметить, что RAND() в конструкции WHERE вычисляется заново при каждом выполнении WHERE.

ROUND(число [, точность])

Эта *функция* округляет число с плавающей запятой до целого числа или, если указан второй *аргумент*, до заданного количества цифр после запятой. Если *точность* отрицательная, обнуляется целая часть числа.

На рис. 24(а), 24(б), 24(в), 24(г), 24(д) и 24(е) приведены примеры работы с функцией ROUND.

```
mysql> SELECT ROUND(-1.23);
+-------------------------+
|       ROUND(-1.23)      |
+-------------------------+
|           -1            |
+-------------------------+
1 row in set (0.00 sec)
```

Рис. 24(а). Округление числа

```
mysql> SELECT ROUND(-1.58);
+-------------------------+
|       ROUND(-1.58)      |
+-------------------------+
|           -2            |
+-------------------------+
1 row in set (0.00 sec)
```

Рис. 24(б). Округление числа

```
mysql> SELECT ROUND(1.58);
+-------------------------+
|       ROUND(1.58)       |
+-------------------------+
|           2             |
+-------------------------+
1 row in set (0.00 sec)
```

Рис. 24(в). Округление числа

```
mysql> SELECT ROUND(1.298, 1);
+-------------------------+
|     ROUND(1.298, 1)     |
+-------------------------+
|           1.3           |
+-------------------------+
1 row in set (0.00 sec)
```

Рис. 24(г). Округление числа

```
mysql> SELECT ROUND(1.298, 0);
+-------------------------+
|     ROUND(1.298, 0)     |
+-------------------------+
|           1             |
+-------------------------+
1 row in set (0.00 sec)
```

Рис. 24(д). Округление числа

```
mysql> SELECT ROUND(23.298, -1);
+-----------------------+
|   ROUND(23.298, -1)   |
+-----------------------+
|           20          |
+-----------------------+
1 row in set (0.00 sec)
```

Рис. 24(е). Округление числа

Следует отметить, что поведение ROUND(), когда *аргумент* точно на середине отрезка между двумя целыми, зависит от реализации библиотеки C. Различные реализации округляют до ближайшего четного, либо всегда в большую сторону, либо всегда в меньшую сторону, либо в сторону ближайшего нуля. Если вам нужно иметь предсказуемое поведение в этом случае, применяйте вместо этой функции TRUNCATE()ИЛИ FLOOR().

SIGN(число)

Возвращает знак аргумента как -1, 0 или 1, в зависимости от того, число отрицательное, нуль или положительное.

На рис. 25(а), 25(б) и 25(в) приведены примеры работы с функцией SIGN.

```
mysql> SELECT SIGN(-32);
+-----------------------+
|      SIGN(-32)        |
+-----------------------+
|           -1          |
+-----------------------+
1 row in set (0.00 sec)
```

Рис. 25(а). Знак числа

```
mysql> SELECT SIGN(0);
+----------------------+
|        SIGN(0)       |
+----------------------+
|          0           |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 25(б). Знак числа

```
mysql> SELECT SIGN(234);
+----------------------+
|        SIGN(234)     |
+----------------------+
|           1          |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 25(в). Знак числа

SIN(число)

Эта *функция* возвращает синус числа в радианах.

На рис. 26(а) и 26(б) приведены примеры работы с функцией SIN.

```
mysql> SELECT SIN(1);
+-----------------------+
|         SIN(1)        |
+-----------------------+
|        0.841471       |
+-----------------------+
1 row in set (0.00 sec)
```

Рис. 26(а). Синус числа

```
mysql> SELECT SIN(PI());
+-----------------------+
|       SIN(PI())       |
+-----------------------+
|        0.000000       |
+-----------------------+
1 row in set (0.00 sec)
```

Рис. 26(б). Синус числа

SQRT(число)

Эта *функция* возвращает квадратный корень числа.

На рис. 27(а), 27(б) и 27(в) приведены примеры работы с функцией SQRT.

```
mysql> SELECT SQRT(15);
+-----------------------+
|       SQRT(15)        |
+-----------------------+
|       3.872983        |
+-----------------------+
1 row in set (0.00 sec)
```

Рис. 27(а). Квадратный корень

```
mysql> SELECT SQRT(4);
+-----------------------+
|        SQRT(4)        |
+-----------------------+
|        2.000000       |
+-----------------------+
1 row in set (0.00 sec)
```

Рис. 27(б). Квадратный корень

```
mysql> SELECT SQRT(20);
+-----------------------+
|        SQRT(20)       |
+-----------------------+
|        4.472136       |
+-----------------------+
1 row in set (0.00 sec)
```

Рис. 27(в). Квадратный корень

TAN(число)

Возвращает тангенс числа.

На рис. 28 приведен пример работы с функцией TAN.

```
mysql> SELECT TAN(PI()+1);
+----------------------+
|      TAN(PI()+1)     |
+----------------------+
|        1.557408      |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 28. Тангенс числа

TRUNCATE(число1, число2)

Возвращает число1 с дробной частью, усеченной до число2 десятичных разрядов. Если число2 равно 0, результат не имеет точки и дробной части. Если число2 отрицательное, целая часть числа длиночисло2 обнуляется.

На рис. 29(а), 29(б), 29(в), 29(г) и 29(д) приведены примеры работы с функцией TRUNCATE.

```
mysql> SELECT TRUNCATE(1.223,1);
+----------------------+
|   TRUNCATE(1.223,1)  |
+----------------------+
|          1.2         |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 29(а). Усечение числа

```
mysql> SELECT TRUNCATE(1.999,1);
+---------------------------+
|      TRUNCATE(1.999,1)    |
+---------------------------+
|            1.9            |
+---------------------------+
1 row in set (0.00 sec)
```

Рис. 29(б). Усечение числа

```
mysql> SELECT TRUNCATE(1.999,0);
+----------------------+
|   TRUNCATE(1.999,0)  |
+----------------------+
|          1           |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 29(в). Усечение числа

```
mysql> SELECT TRUNCATE(-1.999,1);
+-----------------------+
|   TRUNCATE(-1.999,1)  |
+-----------------------+
|           -1.9        |
+-----------------------+
1 row in set (0.00 sec)
```

Рис. 29(г). Усечение числа

```
mysql> SELECT TRUNCATE(122,-2);
+-----------------------+
|   TRUNCATE(122,-2)    |
+-----------------------+
|           100         |
+-----------------------+
1 row in set (0.00 sec)
```

Рис. 29(д). Усечение числа

Все числа округляются в сторону нуля. Следует отметить, что десятичные числа обычно не хранятся в компьютерах именно в виде чисел, а в виде двоичных значений двойной точности, поэтому иногда результат может вызвать удивление (рис. 29(е))

```
mysql> SELECT TRUNCATE(10.28*100,0);
+---------------------------+
|   TRUNCATE(10.28*100,0)   |
+---------------------------+
|           1027            |
+---------------------------+
1 row in set (0.00 sec)
```

Рис. 29(е). Усечение числа

Это происходит потому, что 10.28 на самом деле сохраняется как 10.27999999999999...

Часть 2 Работа с датой и временем

Цель занятия: формирование навыка работы с запросами, с применением даты и времени

Этапы выполнения работы:

1. Запустить MySql.
2. Выбрать БД employees.
3. Создать запросы, указанные ниже.
4. Продемонстрировать полученные запросы преподавателю, защитить выполненную работу.

ХОД РАБОТЫ

Работа с датой

Создать еще одну таблицу, чтобы понять *тип данных* date (дата).

Создадим в текстовом редакторе *файл* employee_per.dat, который содержит оператор создания таблицы CREATE следующего вида:

```
CREATE TABLE employee_per (
e_id int unsigned not null primary key auto_incremebt,  -- идентификационный
номер
address varchar(60),                    -- адрес
phone varchar(11),                      -- номер телефона
p_email varchar(60),                    -- e-mail
birth_date DATE,                        -- дата рождения
sex ENUM('M', 'Ж'),                     -- пол
m_status ENUM('Y','N'),                 -- статус
s_name varchar(40),                     -- имя
children int);                          -- количество детей
```

и *последовательность операторов* INSERT, например, такого вида. Количество записей может быть произвольно.

```
INSERT INTO employee_per (address, phone, p_email, birth_date, sex, m_status,
s_name, children) values ('Красноармейская, 154', 89286548596, 'a67456@yandex.ru',
'1986-04-12', 'Ж', 'Y', 'Анна Серегина', 1);
```

Затем загрузим этот *файл*, как мы делали раньше, в базу данных.

В системе *Windows*

1) Поместите *файл* в каталог c:*mysql*\bin.

2) Выполните в приглашении *DOS* команду.

```
dosprompt> mysql employees <employee_per.dat
```

3) Запустите программу клиента mysql и проверьте, что таблица была создана, с помощью команды SHOW TABLES;

4) Данные таблицы можно вывести с помощью команды DESCRIBE.

```
mysql> DESCRIBE employee_per;
```

e_id: *идентификатор* сотрудника, такой же как в таблице employee_data
address: *адрес* сотрудника
phone: номер телефона
p_email: личный *адрес e-mail*
birth_date: дата рождения
sex: Пол сотрудника, мужской (M) или женский (Ж)
m_status: семейное положение, в браке (Y) или холост (N).
s_name: Имя супруга (NULL, если сотрудник холост)
children: Число детей (NULL, если детей нет)

Теоретические сведения

1 Особенности типа данных Date

Даты в MySQL всегда представлены с годом, за которым следует месяц и затем день месяца. Даты часто записывают в виде YYYY-MM-DD, где YYYY -- 4 цифры года, MM -- 2 цифры месяца и DD -- 2 цифры дня месяца.

2 Операции с датами

Тип столбца даты позволяет выполнять несколько операций, таких как сортировка, проверка условий с помощью операторов сравнения и т.д.

3 Использование операторов = и !=

Поиск по дате рождения:

```
select p_email, phone from employee_per where birth_date = '1992-12-31';
```

Примечание: MySQL требует, чтобы даты были заключены в кавычки.

4 Использование операторов >= и <=

Поиск по дате рождения с использованием оператора >=:

```
select e_id, birth_date from employee_per where birth_date >= '1979-01-01';
```

5 Определение диапазонов

```
select e_id, birth_date
from employee_per where
birth_date BETWEEN
'1992-01-01' AND '1999-01-01';
```

Тот же запрос можно представить без конструкции BETWEEN:

```
select e_id, birth_date
from employee_per where
birth_date >= '1969-01-01' AND birth_date <= '1974-01-01';
```

6 Использование Date для сортировки данных

Поиск по дате рождения в определенном диапазоне:

```
select e_id, birth_date
from employee_per
ORDER BY birth_date;
```

7 Выбор данных с помощью Date

Вот как можно выбрать сотрудников, которые родились в марте.

```
select e_id, birth_date
from employee_per
where MONTH(birth_date) = 3;
```

Можно также использовать вместо чисел названия месяцев.

```
select e_id, birth_date
from employee_per
where MONTHNAME(birth_date) = 'January';
```

Поиск по году рождения:

```
 select e_id, birth_date
from employee_per
where year(birth_date) = 1972;
```

Поиск по дате рождения

```
select e_id, birth_date
from employee_per
where DAYOFMONTH(birth_date) = 20;
```

8 Текущие даты

Поиск по текущему месяцу

```
select e_id, birth_date
from employee_per where
MONTH(birth_date) = MONTH(CURRENT_DATE);
```

9 Тип столбца Null

Тип столбца NULL является специальным значением. Чтобы вставить *значение* NULL, удалите просто имя столбца из оператора INSERT. Столбцы содержат NULL по умолчанию, если только не определены как NOT NULL. *Значение* null может использоваться для целочисленных, а также текстовых или двоичных данных.

NULL нельзя сравнивать с помощью арифметических операторов. Сравнение для NULL можно делать с помощью IS NULL или IS NOT NULL.

Сотрудники, имеющие детей

```
select e_id, children
from employee_per
where children IS NOT NULL;
```

Задания

1. Вывести идентификаторы и даты рождения всех сотрудников, которые родились до 1965 г.
2. Вывести идентификаторы и даты рождения сотрудников, родившихся между 1970 и 1973 гг.
3. Вывести идентификаторы, даты рождения и адреса e-mail сотрудников, родившихся в апреле.
4. Вывести идентификаторы, даты рождения и имена супругов сотрудников, родившихся в 1968 г., и отсортируйте записи на основе имен их супругов.
5. Выведите идентификаторы сотрудников, родившихся в текущем месяце.
6. Сколько в базе данных имеется уникальных годов рождения?
7. Вывести список уникальных годов рождения и число сотрудников, родившихся в каждом таком году.
8. Сколько сотрудников родились в каждом месяце? Выдача должна содержать названия месяцев (не номера), и записи должны быть упорядочены по убыванию по месяцам, начиная от наибольшего номера.
9. Найти и вывести идентификаторы и имена супругов всех сотрудников, которые состоят в браке.
10. Изменить предыдущее задание так, чтобы вывод был отсортирован по именам супругов.
11. Сколько имеется сотрудников каждого пола (мужчин и женщин)?
12. Сколько сотрудников состоят в браке, и сколько холостых?
13. Найдите общее число детей.
14. Сделайте уникальные группы по количеству детей и определите число детей каждой группы. Отсортируйте вывод групп по убыванию по количеству детей.

Практическое занятие № 7 Хранимые процедуры

Цель занятия: формирование навыка работы создания процедур

Этапы выполнения работы:

1. Запустить MySql.
2. Выбрать БД magazin.
3. Создать хранимые процедуры, указанные ниже.
4. Продемонстрировать полученные процедуры преподавателю, защитить выполненную работу.

ХОД РАБОТЫ

Хранимые процедуры позволяют объединить последовательность запросов и сохранить их на сервере.

Синтаксис:	`CREATE PROCEDURE имя_процедуры (параметры)` `Begin` `  Операторы` `end`

Параметры это те данные, которые мы будем передавать процедуре при ее вызове, а операторы - это собственно запросы.

```
INSERT INTO customers  (name, email) VALUE ('Иванов Сергей', 'sergo@mail.ru');
```

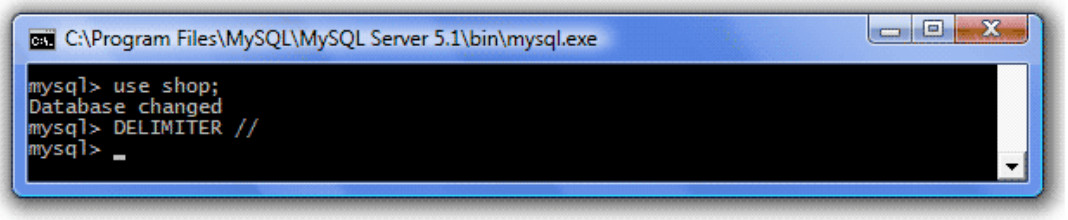
Этот же запрос вполне уместно оформить в виде процедуры:

```
CREATE PROCEDURE ins_cust(n CHAR(50), e CHAR(50))
     begin
        insert into customers  (name, email) value (n, e);
     end
```

Обратите внимание, как задаются параметры: необходимо дать имя параметру и указать его тип, а в теле процедуры уже используем имена параметров.

Один нюанс. Как вы помните, точка с запятой означает конец запроса и отправляет его на выполнение, что в данном случае неприемлемо. Поэтому, прежде, чем написать процедуру необходимо переопределить разделитель с ; на "//", чтобы запрос не отправлялся раньше времени. Делается это с помощью оператора DELIMITER //:

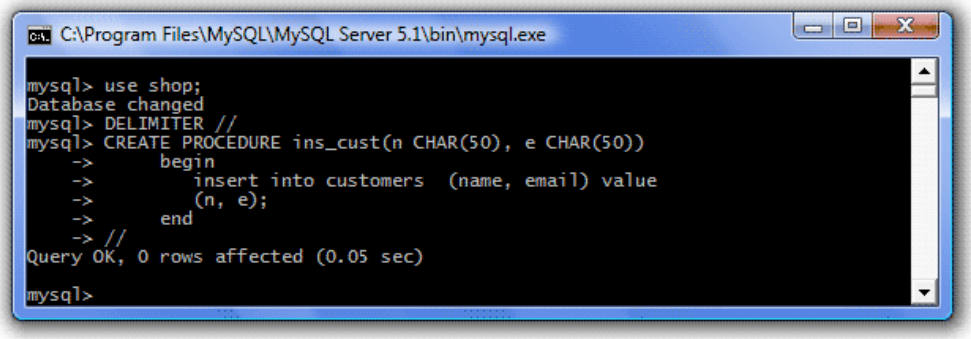
```
DELIMITER //
```



Таким образом, указали СУБД, что выполнять команды теперь следует после //. Следует помнить, что переопределение разделителя осуществляется только на один сеанс работы, т.е. при

следующем сеансе работы с MySql разделитель снова станет точкой с запятой и при необходимости его придется снова переопределять
Теперь можно разместить процедуру:

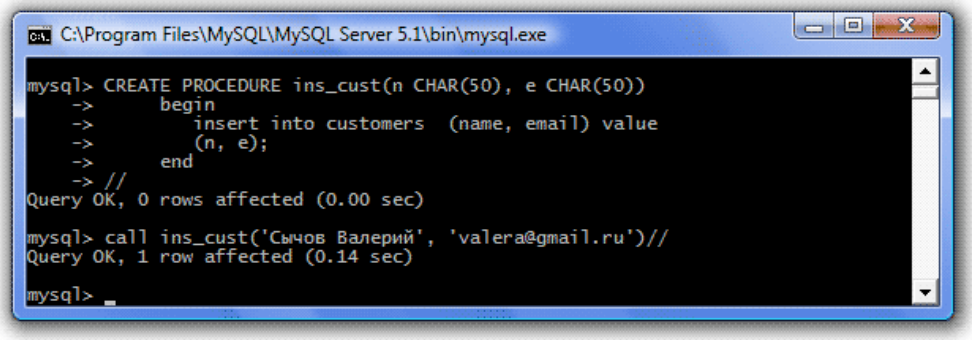
```
CREATE PROCEDURE ins_cust(n CHAR(50), e CHAR(50))
     begin
        insert into customers  (name, email) value (n, e);
     end
//
```



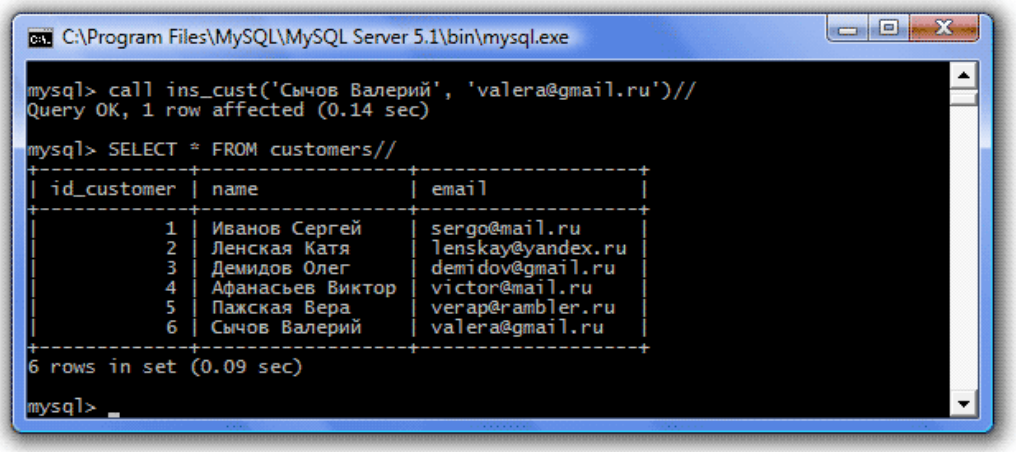
Итак, процедура создана. Теперь, когда нам понадобится ввести нового покупателя нам достаточно ее вызвать, указав необходимые параметры. Для вызова хранимой процедуры используется оператор CALL, после которого указывается имя процедуры и ее параметры.

Добавьте нового покупателя в таблицу Покупатели (customers):

```
call ins_cust('Сычов Валерий', 'valera@gmail.ru')//
```



Проверьте, работает ли процедура, посмотрев, появился ли новый покупатель в таблице Покупатели (customers):



Процедура будет работать до тех пор, пока не будет удалена помощью оператора DROP PROCEDURE название_процедуры.

Процедуры позволяют объединить последовательность запросов. Чтобы проверить, на какую сумму привез товар поставщик "Дом печати", можно было бы взять уже написанные ранее представление и запрос к нему, объединить в хранимую процедуру и сделать идентификатор поставщика (id_vendor) входным параметром, вот так:

```
CREATE PROCEDURE sum_vendor(i INT)
     begin
        CREATE VIEW report_vendor AS SELECT magazine_incoming.id_product,
magazine_incoming.quantity,
          prices.price, magazine_incoming.quantity*prices.price AS summa FROM
magazine_incoming, prices
        WHERE magazine_incoming.id_product= prices.id_product AND id_incoming=
          (SELECT id_incoming FROM incoming WHERE id_vendor=i);
        SELECT SUM(summa) FROM report_vendor;
     end
//
```

Но так процедура работать не будет. Все дело в том, что в представлениях не могут использоваться параметры. Поэтому придется несколько изменить последовательность запросов.

Сначала создайте представление, которое будет выводить идентификатор поставщика (id_vendor), идентификатор продукта (id_product), количество (quantity), цену (price) и сумму (summa) из трех таблиц Поставки (incoming), Журнал поставок (magazine_incoming), Цены (prices):

```
CREATE VIEW report_vendor AS SELECT incoming.id_vendor,
  magazine_incoming.id_product, magazine_incoming.quantity,
  prices.price, magazine_incoming.quantity*prices.price AS summa
FROM incoming, magazine_incoming, prices
WHERE magazine_incoming.id_product= prices.id_product AND
  magazine_incoming.id_incoming=                    incoming.id_incoming;
```

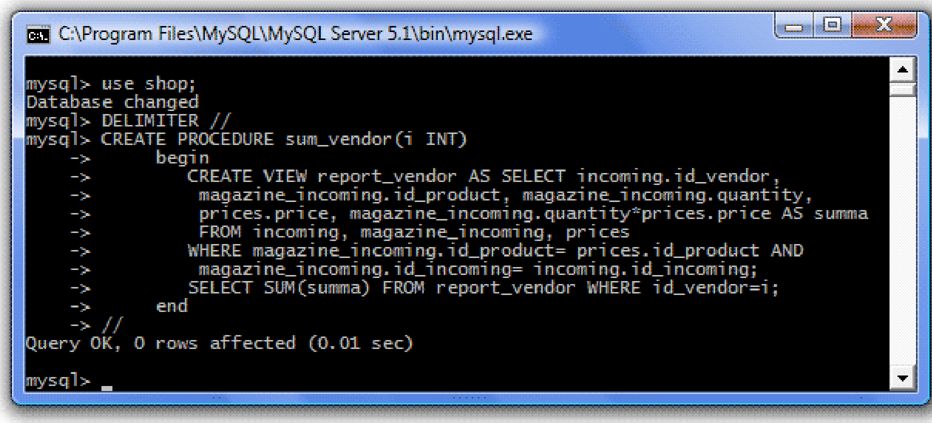
Затем запрос, который просуммирует суммы поставок интересующего поставщика, например, с id_vendor=2:

```
SELECT SUM(summa) FROM report_vendor WHERE id_vendor=2;
```

Теперь объедините два этих запроса в хранимую процедуру, где входным параметром будет

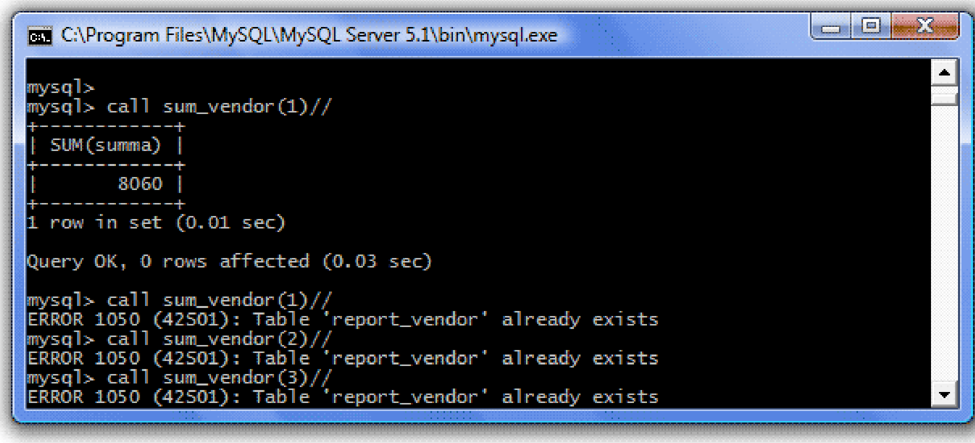
идентификатор поставщика (id_vendor), который будет подставляться во второй запрос, но не в представление:

```
CREATE PROCEDURE sum_vendor(i INT)
     begin
        CREATE VIEW report_vendor AS SELECT incoming.id_vendor,
           magazine_incoming.id_product, magazine_incoming.quantity,
           prices.price, magazine_incoming.quantity*prices.price AS summa
           FROM incoming, magazine_incoming, prices
           WHERE magazine_incoming.id_product= prices.id_product AND
           magazine_incoming.id_incoming= incoming.id_incoming;
        SELECT SUM(summa) FROM report_vendor WHERE id_vendor=i;
     end
//
```



Проверьте работу процедуры, с разными входными параметрами:

```
call sum_vendor(1)//
call sum_vendor(2)//
call sum_vendor(3)//
```

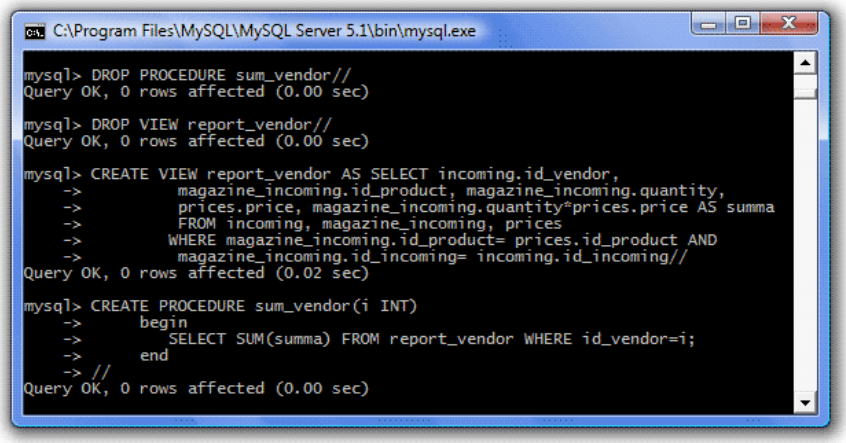


Процедура срабатывает один раз, а затем выдает ошибку, говоря, что представление report_vendor уже имеется в БД.

Так происходит потому, что при обращении к процедуре в первый раз, она создает представление. При обращении во второй раз, она снова пытается создать представление, но оно уже есть, поэтому и появляется ошибка. Чтобы избежать этого возможно два варианта.

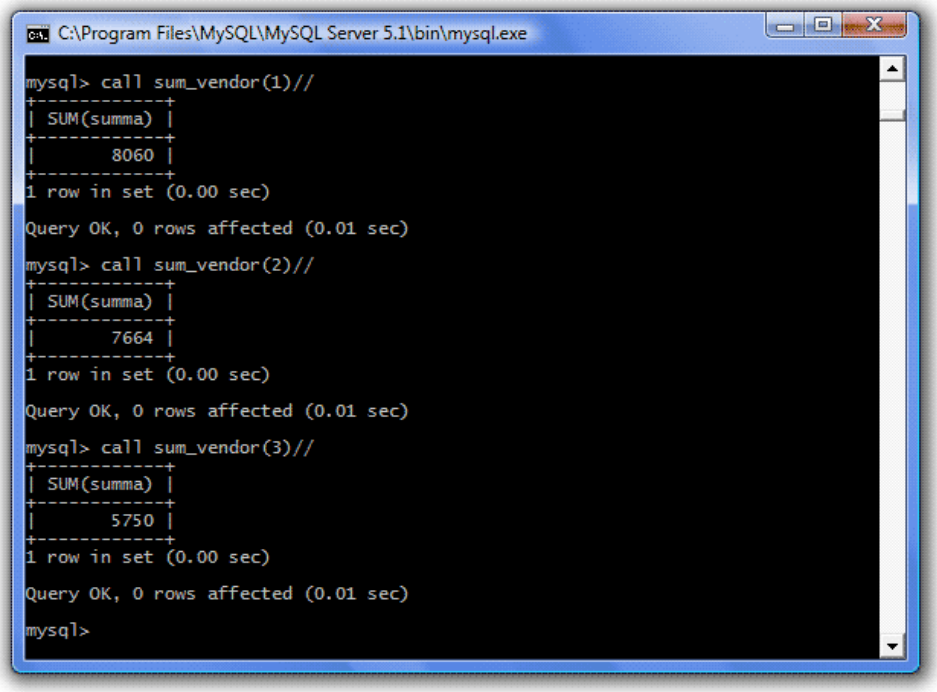
Первый - вынести представление из процедуры. То есть один раз создать представление, а процедура будет лишь к нему обращаться, но не создавать его. Предварительно не забудет удалить уже созданную процедуру и представление:

```
DROP PROCEDURE sum_vendor//
DROP VIEW report_vendor//
CREATE VIEW report_vendor AS SELECT incoming.id_vendor,
     magazine_incoming.id_product, magazine_incoming.quantity,
     prices.price, magazine_incoming.quantity*prices.price AS summa
     FROM incoming, magazine_incoming, prices
     WHERE magazine_incoming.id_product= prices.id_product AND
     magazine_incoming.id_incoming= incoming.id_incoming//
CREATE PROCEDURE sum_vendor(i INT)
     begin
        SELECT SUM(summa) FROM report_vendor WHERE id_vendor=i;
     end
//
```



Проверяем работу:

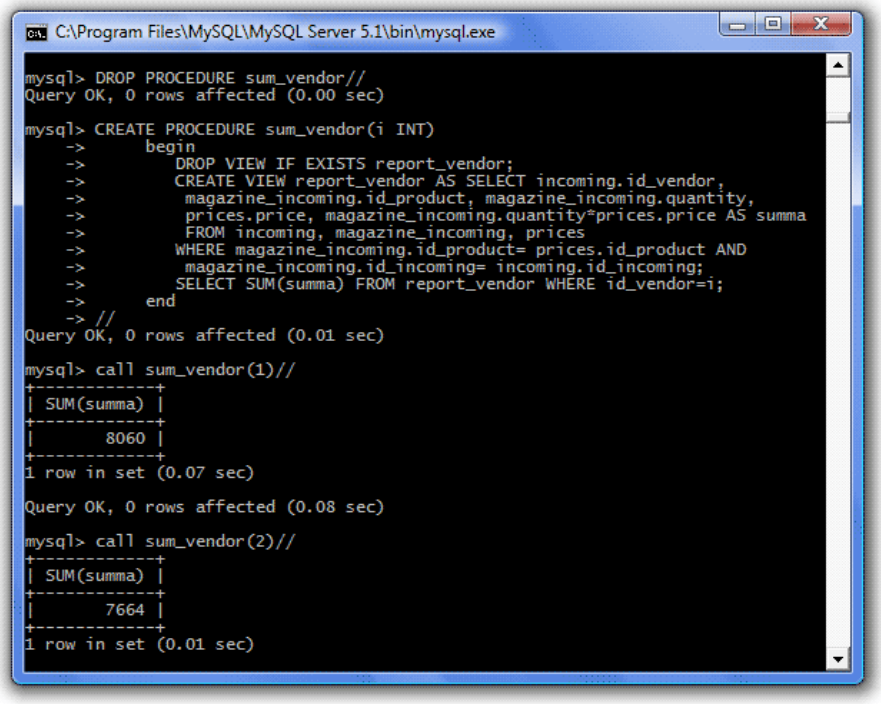
```
call sum_vendor(1)//
call sum_vendor(2)//
call sum_vendor(3)//
```



Второй вариант - прямо в процедуре дописать команду, которая будет удалять представление, если оно существует:

```
CREATE PROCEDURE sum_vendor(i INT)
     begin
        DROP VIEW IF EXISTS report_vendor;
        CREATE VIEW report_vendor AS SELECT incoming.id_vendor,
          magazine_incoming.id_product, magazine_incoming.quantity,
          prices.price, magazine_incoming.quantity*prices.price AS summa
          FROM incoming, magazine_incoming, prices
          WHERE magazine_incoming.id_product= prices.id_product AND
          magazine_incoming.id_incoming= incoming.id_incoming;
        SELECT SUM(summa) FROM report_vendor WHERE id_vendor=i;
     end
//
```

Перед использованием этого варианта не забудьте удалить процедуру sum_vendor, а затем проверить работу:



Чтобы проверить, какие хранимые процедуры имеются на сервере, и как они выглядят

- SHOW PROCEDURE STATUS// - позволяет просмотреть список имеющихся хранимых процедур. Правда просматривать этот список не очень удобно, т.к. по каждой процедуре выдается информация об имени БД, к которой процедура принадлежит, ее типе, учетной записи, от имени которой была создана процедура, о дате создания и изменения процедуры и т.д. И все-таки, если вам необходимо посмотреть, какие процедуры у вас есть, то стоит воспользоваться этим оператором.

- SHOW CREATE PROCEDURE имя_процедуры// - позволяет получить информацию о конкретной процедуре, в частности просмотреть ее код. Вид для просмотра также не очень удобный, но разобраться можно.

В системной базе данных MySQL есть таблица proc, где и хранится информация о процедурах. Можно сделать SELECT-запрос к этой таблице. Причем, создав привычный запрос:

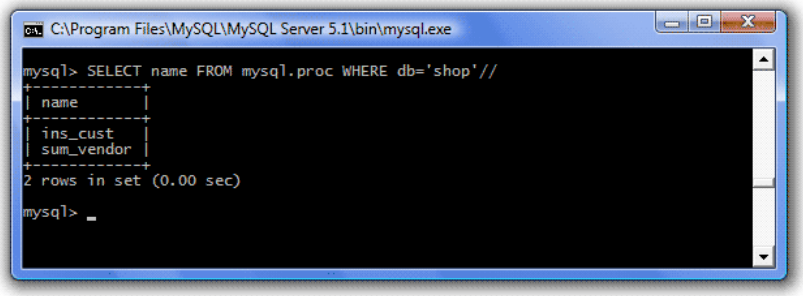
```
SELECT * FROM mysql.proc//
```

Получите нечто такое же нечитабельное, как и при использовании операторов SHOW. Поэтому нужно создавать запросы с условиями. Например, если создать такой запрос:

```
SELECT name FROM mysql.proc//
```

То получим имена всех процедур всех баз данных, имеющихся на сервере. Нас, например, на данный момент интересуют только процедуры базы данных shop, поэтому изменим запрос:

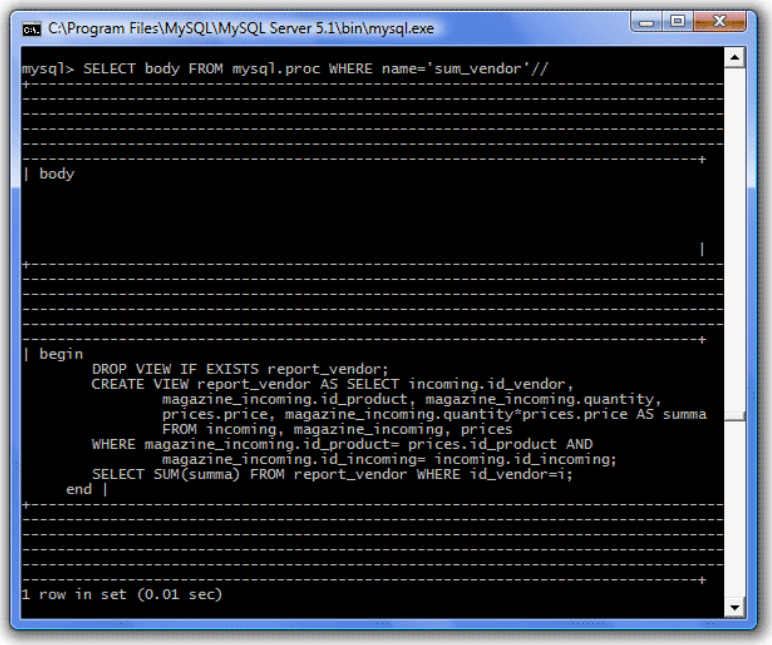
```
SELECT name FROM mysql.proc WHERE db='shop'//
```



Если нужно посмотреть только тело конкретной процедуры (т.е. от begin до end)

```
SELECT body FROM mysql.proc WHERE name='sum_vendor'//
```

И увидим вполне читабельный вариант:



Чтобы извлекать из таблицы proc необходимую информацию, надо просто знать, какие столбцы она содержит, а для этого можно воспользоваться оператором describe имя_таблицы, в данном случае describe mysql.proc.

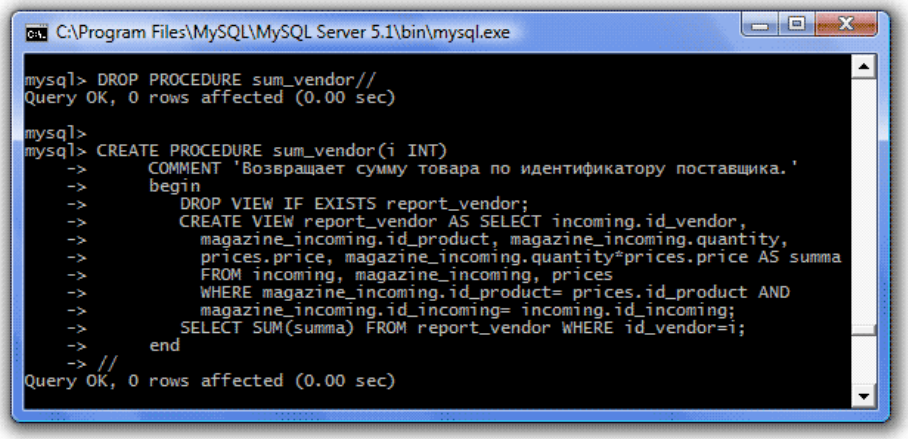
Ниже значения наиболее востребованных столбцов.

- db - имя БД, в которую сохранена процедура.
- name - имя процедуры.
- param_list - список параметров процедуры.
- body - тело процедуры.
- comment - комментарий к хранимой процедуре.

Чтобы создать комментарий, укажите ключевое слово COMMENT 'здесь комментарий'.

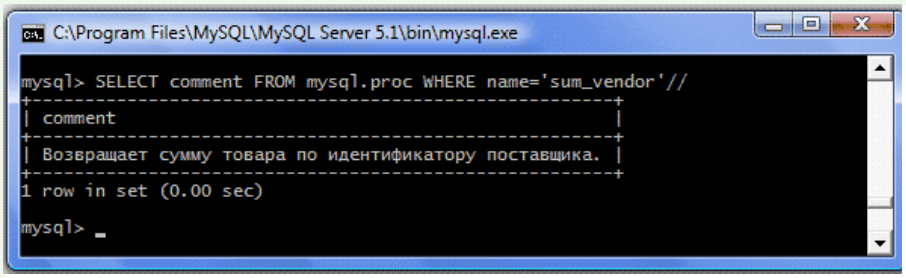
```
CREATE PROCEDURE sum_vendor(i INT)
     COMMENT 'Возвращает сумму товара по идентификатору поставщика.'
```

```
    begin
       DROP VIEW IF EXISTS report_vendor;
       CREATE VIEW report_vendor AS SELECT incoming.id_vendor,
         magazine_incoming.id_product, magazine_incoming.quantity,
         prices.price, magazine_incoming.quantity*prices.price AS summa
         FROM incoming, magazine_incoming, prices
         WHERE magazine_incoming.id_product= prices.id_product AND
         magazine_incoming.id_incoming= incoming.id_incoming;
       SELECT SUM(summa) FROM report_vendor WHERE id_vendor=i;
    end
//
```



А теперь сделаем запрос к комментарию процедуры:

```
SELECT comment FROM mysql.proc WHERE name='sum_vendor'//
```

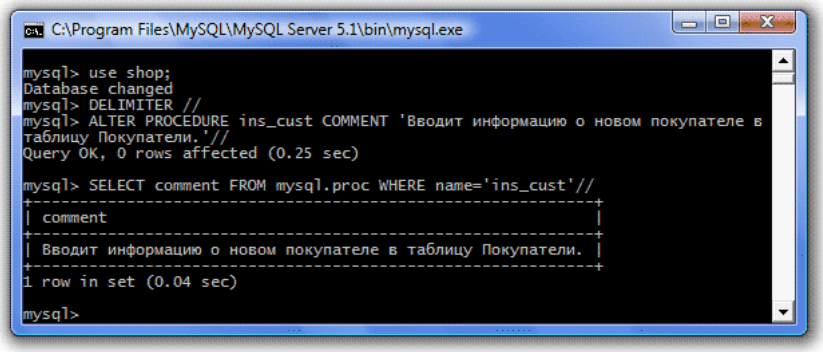


Можно отредактировать имеющуюся хранимую процедуру с помощью оператора ALTER PROCEDURE.

```
ALTER PROCEDURE ins_cust COMMENT 'Вводит информацию о новом покупателе в таблицу Покупатели.'//
```

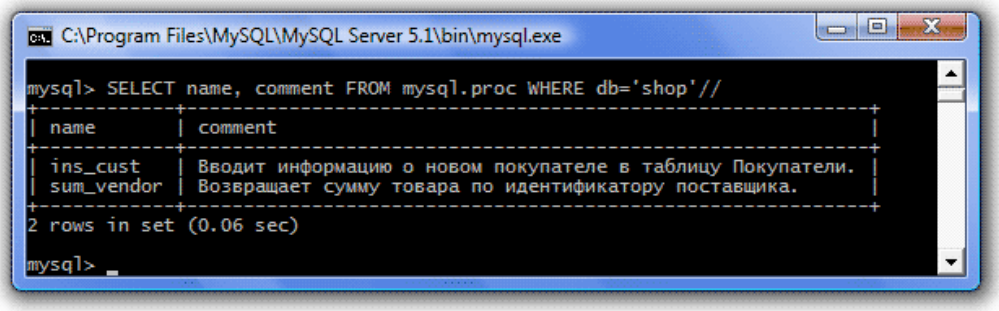
И сделаем запрос к комментарию, чтобы проверить:

```
SELECT comment FROM mysql.proc WHERE name='ins_cust'//
```



Запрос для просмотра комментариев:

```
SELECT name, comment FROM mysql.proc WHERE db='shop'//
```



Практическое занятие № 8 Хранимые процедуры

Цель занятия: формирование навыка работы с хранимыми процедурами, конкретно с оператором IF, с оператором WHILE, REPEAT, LOOP

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Оператор IF...THEN...ELSE	CREATE PROCEDURE имя_процедуры (параметры) begin IF(условие) THEN запрос 1; ELSE запрос 2; END IF; end//	если условие истинно, то выполняется запрос 1, в противном случае - запрос 2
ПЕРЕМЕННАЯ	Объявление переменной начинается с символа собачки (@), за которой следует имя переменной. Объявляются они при помощи оператора SET.	Переменные позволяют сохранить результат текущего запроса для использования в следующих запросах. Действуют только в рамках одного сеанса соединения с сервером MySQL.

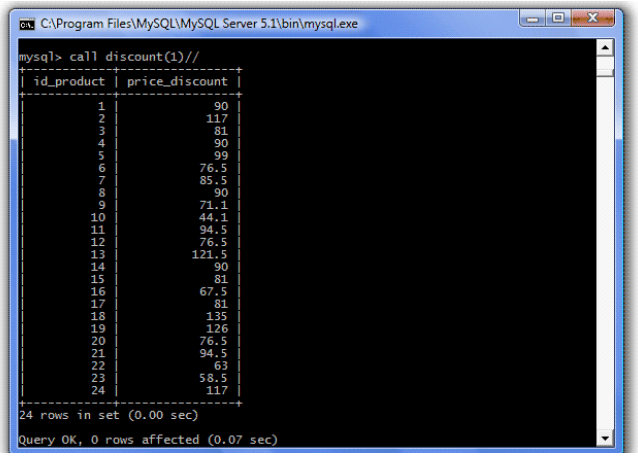
ХОД РАБОТЫ

Предположим, каждый день мы устраиваем в нашем магазине счастливые часы, т.е. делаем скидку 10% на все книги в последний час работы магазина. Чтобы иметь возможность выбирать цену книги, нам необходимо иметь два ее варианта - со скидкой и без. Для этого, нам понадобится создать хранимую процедуру с оператором ветвления. Так как мы имеем всего два варианта цены, то удобнее в качестве входящего параметра иметь булево значение, которое, как вы помните, может принимать либо 0 - ложь, либо 1 - истина. Код процедуры может быть таким:

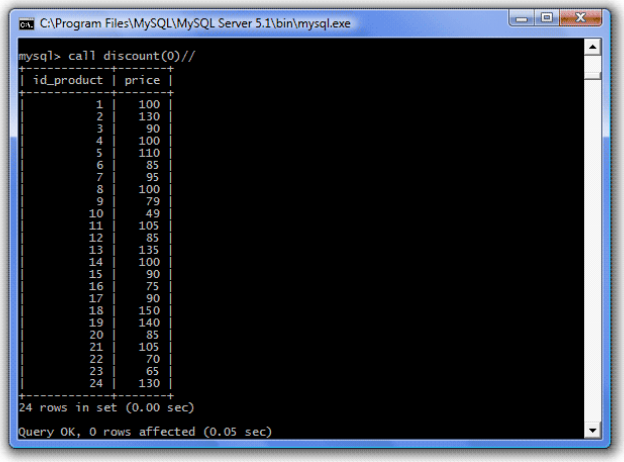
```
CREATE PROCEDURE discount (dis BOOLEAN)
  begin
    IF(dis=1) THEN
       SELECT id_product, price*0.9 AS price_discount FROM prices;
    ELSE
       SELECT id_product, price FROM prices;
    END IF;
  end
      //
```

Т.е. на входе у нас параметр, который может являться, либо 1 (если скидка есть), либо 0 (если скидки нет). В первом случае будет выполнен первый запрос, во втором - второй.

```
call discount(1)//
```




```
call discount(0)//
```



Оператор IF позволяет выбирать и большее количество вариантов запросов, в таком случае используется следующий синтаксис:

```
CREATE PROCEDURE имя процедуры (параметры)
  begin
    IF(условие) THEN
       запрос 1;
    ELSEIF(условие) THEN
       запрос 2;
    ELSE
       запрос 3;
    END IF;
  end
//
```

Причем блоков ELSEIF может быть несколько.
Предположим, что мы решили делать скидки нашим покупателям в зависимости от суммы покупки, до 1000 рублей скидки нет, от 1000 до 2000 рублей - скидка 10%, более 2000 рублей - скидка 20%. Входным параметром для такой процедуры должна быть сумма покупки. Поэтому сначала нам надо написать процедуру, которая будет ее подсчитывать. Сделаем это по аналогии с процедурой sum_vendor, созданной в уроке 15, которая подсчитывала сумму товара по идентификатору поставщика.

Необходимые данные хранятся в двух таблицах Журнал покупок (magazine_sales) и Цены (prices).

```
CREATE PROCEDURE sum_sale(IN i INT)
     COMMENT 'Возвращает сумму покупки по ее идентификатору.'
     begin
        DROP VIEW IF EXISTS sum_sale;
        CREATE VIEW sum_sale AS SELECT magazine_sales.id_sale,
          magazine_sales.id_product, magazine_sales.quantity,
          prices.price, magazine_sales.quantity*prices.price AS summa
          FROM magazine_sales, prices
          WHERE magazine_sales.id_product=prices.id_product;
        SELECT SUM(summa) FROM sum_sale WHERE id_sale=i;
     end
//
```

Здесь перед параметром появилось новое ключевое слово IN. Дело в том, что мы можем, как передавать данные в процедуру, так и передавать данные из процедуры. По умолчанию, т.е. если опустить слово IN, параметры считаются входными (поэтому раньше слово не использовали). Здесь же мы явно указали, что параметр i является входным. Если же нам понадобится извлечь какие-нибудь данные из хранимой процедуры, то мы будем использовать ключевое слово OUT, но об этом чуть позже.

Теперь напишите процедуру, которая пересчитает итоговую сумму с учетом предоставляемой скидки.
Здесь нам и понадобится оператор ветвления:

```
CREATE PROCEDURE sum_discount(IN sm INT, IN i INT)
     COMMENT 'Возвращает сумму покупки с учетом скидки.'
     begin
       IF((sm>=1000) && (sm<2000)) THEN
          SELECT SUM(summa)*0.9  FROM sum_sale WHERE id_sale=i;
       ELSEIF(sm>=2000) THEN
          SELECT SUM(summa)*0.8  FROM sum_sale WHERE id_sale=i;
           ELSE
```

```
              SELECT SUM(summa)  FROM sum_sale WHERE id_sale=i;
       END IF;
  end
//
```

Т.е. передаем процедуре два входных параметра сумму (sm) и идентификатор покупки (i) и в зависимости от того, какая это сумма, выполняется запрос к представлению sum_sale на подсчет итоговой суммы покупки, умноженной на нужный коэффициент.

Осталось только сделать так, чтобы сумма покупки автоматически передавалась в эту процедуру. Для этого процедуру sum_discount вызываем из процедуры sum_sale.

```
CREATE PROCEDURE sum_sale(IN i INT)
     COMMENT 'Возвращает сумму покупки по ее идентификатору.'
     begin
        DROP VIEW IF EXISTS sum_sale;
        CREATE VIEW sum_sale AS SELECT magazine_sales.id_sale,
          magazine_sales.id_product, magazine_sales.quantity,
          prices.price, magazine_sales.quantity*prices.price AS summa
          FROM magazine_sales, prices
          WHERE magazine_sales.id_product=prices.id_product;
        SELECT SUM(summa) FROM sum_sale WHERE id_sale=i;
        CALL sum_discount(?, i);
     end
//
```

Вопросительный знак при вызове процедуры sum_discount поставлен, т.к. не понятно, как результат предыдущего запроса (т.е. итоговой суммы) передать в процедуру sum_discount. Кроме того, не понятно, как процедура sum_discount вернет результат своей работы.

Теперь используем параметр с ключевым словом OUT, т.е. параметр, который будет возвращать данные из процедуры.
Введем такой параметр ss, так как сумма может быть и дробным числом, зададим ему тип DOUBLE:

```
CREATE PROCEDURE sum_discount(IN sm INT, IN i INT, OUT ss DOUBLE)
     COMMENT 'Возвращает сумму покупки с учетом скидки.'
     begin
       IF((sm>=1000) && (sm<2000)) THEN
          SELECT SUM(summa)*0.9  FROM sum_sale WHERE id_sale=i;
       ELSEIF(sm>=2000) THEN
          SELECT SUM(summa)*0.8  FROM sum_sale WHERE id_sale=i;
       ELSE
             SELECT SUM(summa)  FROM sum_sale WHERE id_sale=i;
       END IF;
  end
//

CREATE PROCEDURE sum_sale(IN i INT, OUT ss DOUBLE)
     COMMENT 'Возвращает сумму покупки по ее идентификатору.'
     begin
        DROP VIEW IF EXISTS sum_sale;
        CREATE VIEW sum_sale AS SELECT magazine_sales.id_sale,
          magazine_sales.id_product, magazine_sales.quantity,
          prices.price, magazine_sales.quantity*prices.price AS summa
          FROM magazine_sales, prices
          WHERE magazine_sales.id_product=prices.id_product;
        SELECT SUM(summa)  FROM sum_sale WHERE id_sale=i;
        CALL sum_discount(?, i, ss);
```

```
     end
//
```

вызов процедуры CALL sum_discount(?, i, ss); означает, что передавая два первых параметра, мы ждем возврата третьего параметра в процедуру sum_sale. Чтобы внутри самой процедуры sum_discount присвоить этому параметру какое-либо значение, используется ключевое слово INTO:

```
CREATE PROCEDURE sum_discount(IN sm INT, IN i INT, OUT ss DOUBLE)
     COMMENT 'Возвращает сумму покупки с учетом скидки.'
     begin
       IF((sm>=1000) && (sm<2000)) THEN
          SELECT SUM(summa)*0.9 INTO ss FROM sum_sale WHERE id_sale=i;
       ELSEIF(sm>=2000) THEN
          SELECT SUM(summa)*0.8 INTO ss FROM sum_sale WHERE id_sale=i;
       ELSE
             SELECT SUM(summa) INTO ss FROM sum_sale WHERE id_sale=i;
       END IF;
  end
//
```

С помощью ключевого слова INTO, указали, что результат запроса надо передать в параметр ss.

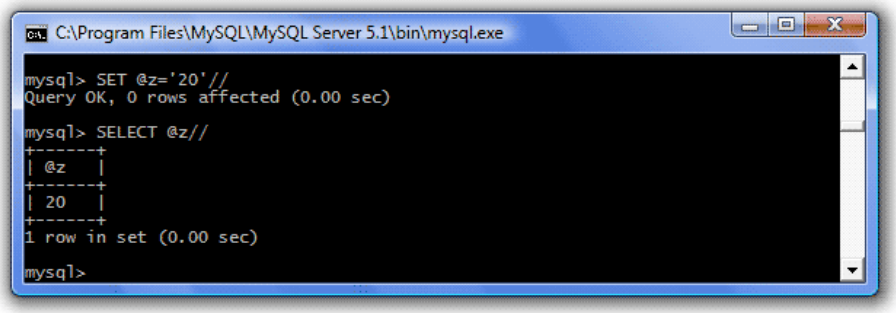
Чтобы передать в процедуру sum_discount результат работы предыдущих запросов.
введем переменную.

Например, объявим переменную z и зададим ей начальное значение 20.

```
SET @z='20'//
```

Переменная с таким значение теперь есть в нашей БД

```
SELECT @z//
```



Для использования переменных в процедурах используется оператор DECLARE, который имеет следующий синтаксис:

```
DECLARE имя_переменной тип DEFAULT значение_по_умолчанию_если_есть
```

Объявим переменную s, в которую будем сохранять значение суммы покупки с помощью ключевого слова INTO:

```
CREATE PROCEDURE sum_sale(IN i INT, OUT ss DOUBLE)
     COMMENT 'Возвращает сумму покупки по ее идентификатору.'
     begin
        DECLARE s INT;
        DROP VIEW IF EXISTS sum_sale;
        CREATE VIEW sum_sale AS SELECT magazine_sales.id_sale,
          magazine_sales.id_product, magazine_sales.quantity,
          prices.price, magazine_sales.quantity*prices.price AS summa
          FROM magazine_sales, prices
          WHERE magazine_sales.id_product=prices.id_product;
        SELECT SUM(summa) INTO s FROM sum_sale WHERE id_sale=i;
        CALL sum_discount(s, i, ss);
     end
//
```

Эта переменная и будет первым входным параметром для процедуры sum_discount

```
CREATE PROCEDURE sum_discount(IN sm INT, IN i INT, OUT ss DOUBLE)
     COMMENT 'Возвращает сумму покупки с учетом скидки.'
     begin
       IF((sm>=1000) && (sm<2000)) THEN
          SELECT SUM(summa)*0.9 INTO ss FROM sum_sale WHERE id_sale=i;
       ELSEIF(sm>=2000) THEN
          SELECT SUM(summa)*0.8 INTO ss FROM sum_sale WHERE id_sale=i;
       ELSE
          SELECT SUM(summa) INTO ss FROM sum_sale WHERE id_sale=i;
       END IF;
  end
//

CREATE PROCEDURE sum_sale(IN i INT, OUT ss DOUBLE)
     COMMENT 'Возвращает сумму покупки по ее идентификатору.'
     begin
        DECLARE s INT;
        DROP VIEW IF EXISTS sum_sale;
        CREATE VIEW sum_sale AS SELECT magazine_sales.id_sale,
          magazine_sales.id_product, magazine_sales.quantity,
          prices.price, magazine_sales.quantity*prices.price AS summa
          FROM magazine_sales, prices
          WHERE magazine_sales.id_product=prices.id_product;
        SELECT SUM(summa) INTO s FROM sum_sale WHERE id_sale=i;
        CALL sum_discount(s, i, ss);
     end
//
```

Алгоритм работы процедуры sum_sale:

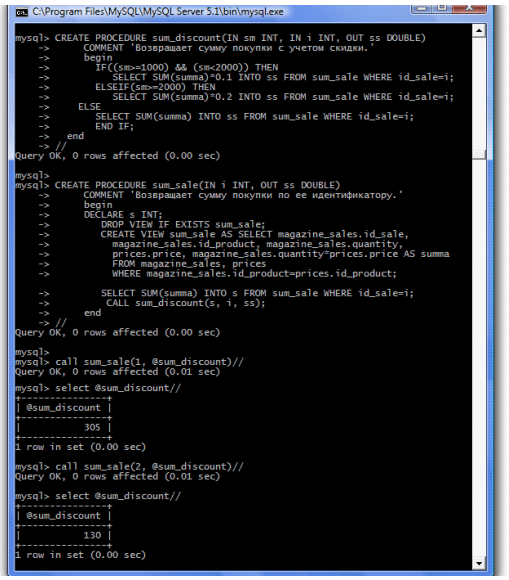
- Вызываем процедуру sum_sale, указывая в качестве входного параметра идентификатор интересующей нас покупки, например id=1, и указывая, что второй параметр - выходной, переменный, являющийся результатом работы процедуры sum_discount:

```
call sum_sale(1, @sum_discount)//
```

- Процедура sum_sale создает представление, в котором собираются данные обо всех покупках, товарах, их количестве, цене и сумме по каждой строчке.

- Затем выполняется запрос к этому представлению на итоговую сумму по покупке с нужным идентификатором, и результат записывается в переменную s.
- Теперь вызывается процедура sum_discount, в которой в качестве первого параметра выступает переменная s (сумма покупки), в качестве второго - идентификатор покупки i, а в качестве третьего указывается параметр ss, который выступает, как выходной, т.е. в него вернется результат действия процедуры sum_discount.
- В процедуре sum_discount проверяется, какому условию соответствует входная сумма, и выполняется соответствующий запрос, результат записывается в выходной параметр ss, который возвращается в процедуру sum_sale.
- Чтобы увидеть результат работы процедуры sum_sale нужно сделать запрос:

```
select @sum_discount//
```



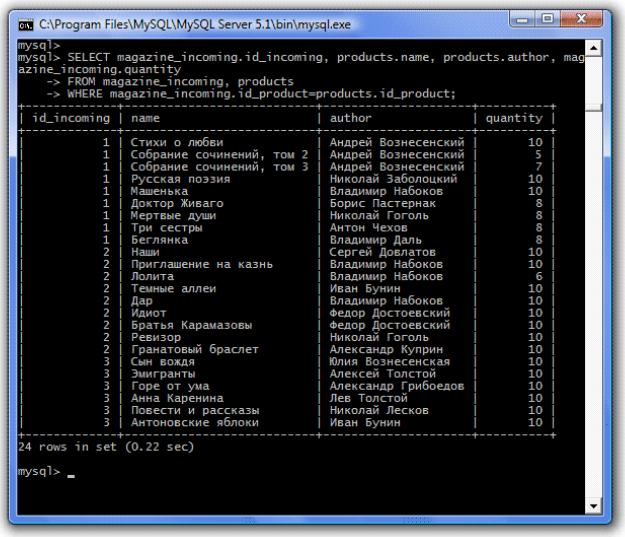
ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Оператор цикла WHILE	`WHILE условие DO` `запрос` `END WHILE`	Запрос будет выполняться до тех пор, пока условие истинно.
Оператор цикла REPEAT	`REPEAT` `запрос` `UNTIL условие` `END REPEAT`	Условие цикла проверяется не в начале, как в цикле WHILE, а в конце, т.е. хотя бы один раз, но цикл выполняется. Сам же цикл выполняется, пока условие ложно
Оператор цикла LOOP	`LOOP` `запрос` `END LOOP`	Этот цикл вообще не имеет условий, поэтому обязательно должен иметь оператор LEAVE.

ХОД УРОКА

Предположим, мы хотим знать названия, авторов и количество книг, которые поступили в различные поставки. Интересующая нас информация хранится в двух таблицах - Журнал Поставок (magazine_incoming) и Товар (products).

```
SELECT magazine_incoming.id_incoming, products.name, products.author,
magazine_incoming.quantity
FROM magazine_incoming, products
WHERE magazine_incoming.id_product=products.id_product;
```



Чтобы результат выводился не в одной таблице, а по каждой поставке отдельно, можно написать 3 разных запроса, добавив в каждый еще одно условие:

```
SELECT magazine_incoming.id_incoming, products.name, products.author,
magazine_incoming.quantity
FROM magazine_incoming, products
WHERE magazine_incoming.id_product=products.id_product AND
magazine_incoming.id_incoming=1;

SELECT magazine_incoming.id_incoming, products.name, products.author,
magazine_incoming.quantity
FROM magazine_incoming, products
WHERE magazine_incoming.id_product=products.id_product AND
magazine_incoming.id_incoming=2;

SELECT magazine_incoming.id_incoming, products.name, products.author,
magazine_incoming.quantity
FROM magazine_incoming, products
WHERE magazine_incoming.id_product=products.id_product AND
magazine_incoming.id_incoming=3;
```

Но гораздо короче сделать это можно с помощью цикла WHILE:

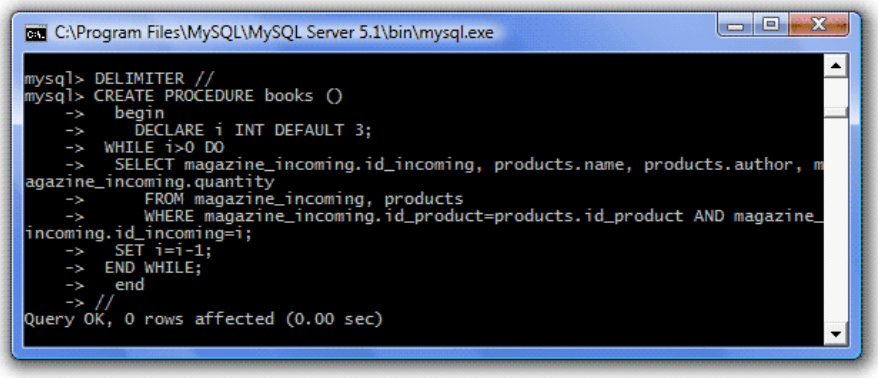
```
DECLARE i INT DEFAULT 3;
   WHILE i>0 DO
     SELECT magazine_incoming.id_incoming, products.name, products.author,
magazine_incoming.quantity
     FROM magazine_incoming, products
     WHERE magazine_incoming.id_product=products.id_product AND
magazine_incoming.id_incoming=i;
     SET i=i-1;
   END WHILE;
```

Ввели переменную i, по умолчанию равную 3, сервер выполнит запрос с id поставки равным 3, затем уменьшит i на единицу (SET i=i-1), убедится, что новое значение переменной i положительно (i>0) и снова выполнит запрос, но уже с новым значением id поставки равным 2.

Так будет происходить, пока переменная i не получит значение 0, условие станет ложным, и цикл закончит свою работу.

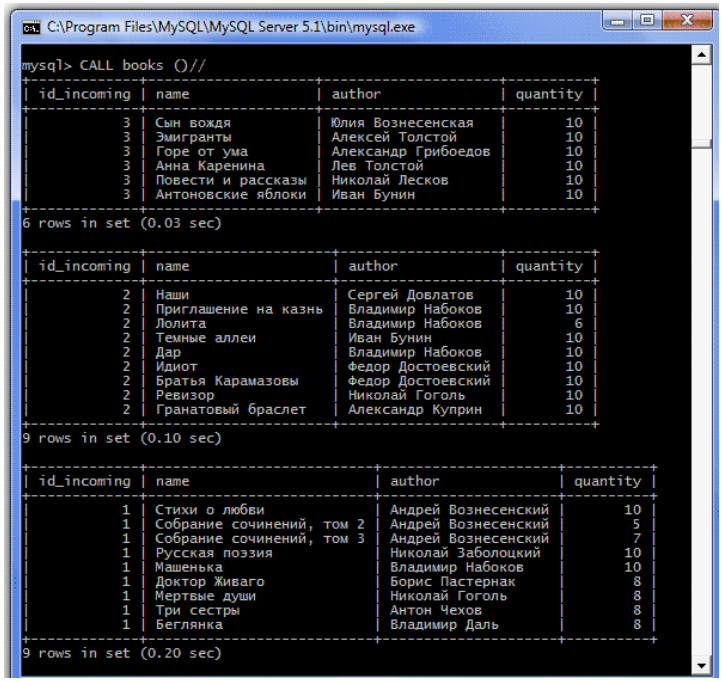
Чтобы убедиться в работоспособности цикла создадим хранимую процедуру books и поместим в нее цикл:

```
DELIMITER //
CREATE PROCEDURE books ()
  begin
    DECLARE i INT DEFAULT 3;
        WHILE i>0 DO
          SELECT magazine_incoming.id_incoming, products.name, products.author,
          magazine_incoming.quantity
          FROM magazine_incoming, products
          WHERE magazine_incoming.id_product=products.id_product
          AND magazine_incoming.id_incoming=i;
          SET i=i-1;
        END WHILE;
  end
//
```



Теперь вызовем процедуру:

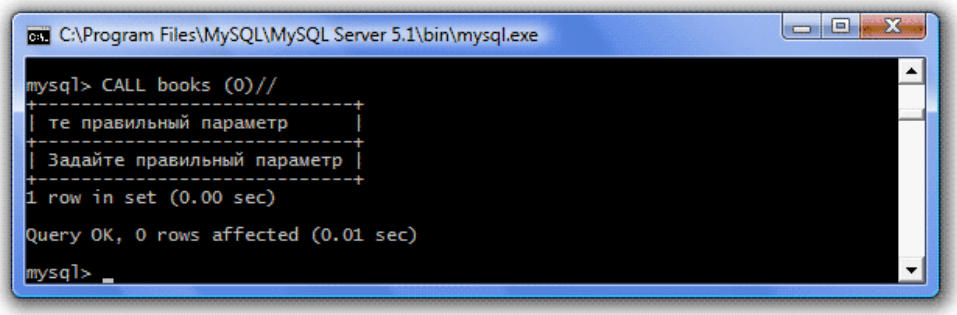
```
CALL books ()//
```



Теперь есть 3 отдельные таблицы (по каждой поставке).

Перепишем процедуру, добавив входной параметр num, и, учитывая, что он не должен быть равен 0:

```
CREATE PROCEDURE books (IN num INT)
  begin
     DECLARE i INT DEFAULT 0;
     IF (num>0) THEN
       WHILE i < num DO
          SELECT magazine_incoming.id_incoming, products.name, products.author,
magazine_incoming.quantity
          FROM magazine_incoming, products
          WHERE magazine_incoming.id_product=products.id_product AND
magazine_incoming.id_incoming=i;
          SET i=i+1;
       END WHILE;
     ELSE
        SELECT 'Задайте правильный параметр';
     END IF;
  end
//
CALL books (0)//
```



У цикла есть один недостаток - если случайно задать слишком большое входное значение, то мы получим псевдобесконечный цикл, который загрузит сервер бесполезной работой. Такие ситуации

предотвращаются с помощью снабжения цикла меткой и использования оператора LEAVE, обозначающего досрочный выход из цикла.

```
CREATE PROCEDURE books (IN num INT)
  begin
    DECLARE i INT DEFAULT 0;
    IF (num>0) THEN
        wet : WHILE i < num DO
          IF (i>10) THEN LEAVE wet;
          ENF IF;
          SELECT magazine_incoming.id_incoming, products.name, products.author,
magazine_incoming.quantity
          FROM magazine_incoming, products
          WHERE magazine_incoming.id_product=products.id_product AND
magazine_incoming.id_incoming=i;
          SET i=i+1;
       END WHILE wet;
    ELSE
        SELECT 'Задайте правильный параметр';
    END IF;
  end
//
```

Добавив в цикл метку wet вначале (wet:) и в конце, а также добавив еще одно условие - если входной параметр больше 10 (число 10 взято произвольно), то цикл с меткой wet следует закончить (IF (i>10) THEN LEAVE wet). Если вызвать процедуру с большим значением num, цикл прервется после 10 итераций (итерация - один проход цикла).

Практическое занятие № 9 Создание триггеров

Цель занятия: формирование навыка работы при создании триггеров

Этапы выполнения работы:

1. Запустить MySql.
2. Выбрать БД magazin.
3. Создать триггеры, указанные ниже.
4. Продемонстрировать полученные результаты преподавателю, защитить выполненную работу.

ХОД РАБОТЫ

Теоретические сведения

Триггер (англ. *trigger*) — это хранимая откомпилированная SQL-процедура, которая не вызывается непосредственно, а исполняется при наступлении определенного события внутри базы данных (вставки, удаления, обновления записей). Поддержка триггеров в MySQL началась с версии 5.0.2

Хранимые процедуры запускают во всех средах, и нет необходимости перестроения логики. С того момента как вы создали хранимую процедуру, не важно какое приложение вы используете для вызова процедуры. Также не важно на каком языке вы программируете, логика процедуры содержится на сервере БД.

Также хранимые процедуры могут сократить сетевой трафик. Сложные, повторяющиеся задачи можно обрабатывать с помощью процедур на сервере Баз данных, без необходимости отсылки промежуточных результатов приложению.

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.

Общий вид синтаксиса для создания тригера:

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name trigger_time trigger_event
    ON tbl_name FOR EACH ROW trigger_body
```

где trigger_name — название триггера;

trigger_time — время срабатывания триггера: BEFORE — перед событием, AFTER — после события;

trigger_event — событие:

- insert — событие возбуждается операторами insert, data load, replace;
- update — событие возбуждается оператором update;
- delete — событие возбуждается операторами delete, replace.

Операторы DROPTABLE и TRUNCATE не активируют выполнение триггера;

tbl_name — название таблицы;

trigger_body — выражение, которое выполняется при активации триггера.

Триггеры могут быть привязаны не к таблице, а к представлению (VIEW). В этом случае с их помощью реализуется механизм «обновляемого представления».

Пример: создадим две таблицы test и log, напишем триггер, который после добавления каждой записи в 1-ю таблицу будет вести лог этого события:

1. Таблица, за которой мы будем следить

```
CREATE TABLE `test` (
`id` INT( 11 ) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
`content` TEXT NOT NULL  );
```

2. Лог

```
CREATE TABLE `log` (
`id` INT( 11 ) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
`msg` VARCHAR( 255 ) NOT NULL,
`time` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
`row_id` INT( 11 ) NOT NULL
    );
```

3. Триггер

```
DELIMITER |
CREATE TRIGGER `update_test` AFTER INSERT ON `test`
FOR EACH ROW BEGIN
   INSERT INTO log Set msg = 'insert', row_id = NEW.id;
END;
```

Здесь оператор DELIMITER служит для определения знака начала/окончания процедуры и может состоять более, чем из одного символа (необходимо выбирать разделитель, который не будет использоваться в процедуре).

На столбцы таблицы, к которой привязан триггер можно ссылаться с помощью псевдонимов OLD и NEW.

OLD.col_name указывает на столбец с именем col_name до изменения или удаления данных.

NEW.col_name относится к колонке новой строке после вставки или существующей - сразу после её обновления.

Для удаления триггера необходимо выполнить запрос:

```
DROP TRIGGER `update_test`;
```

Для просмотра триггеров в базе данных используется оператор:

```
SHOW TRIGGERS;
```

Триггеры имеют несколько важных особенностей использования:

1. Триггеры в MySQL5 могут создаваться только пользователем с привилегией SUPER;
2. При использовании запроса, затрагивающего N - записей, триггер будет запускаться N - раз;
3. После удаления таблицы, СУБД MySQL автоматически удаляет привязанные к ней триггеры.

Практическое занятие №10 Основы работы с dbForge

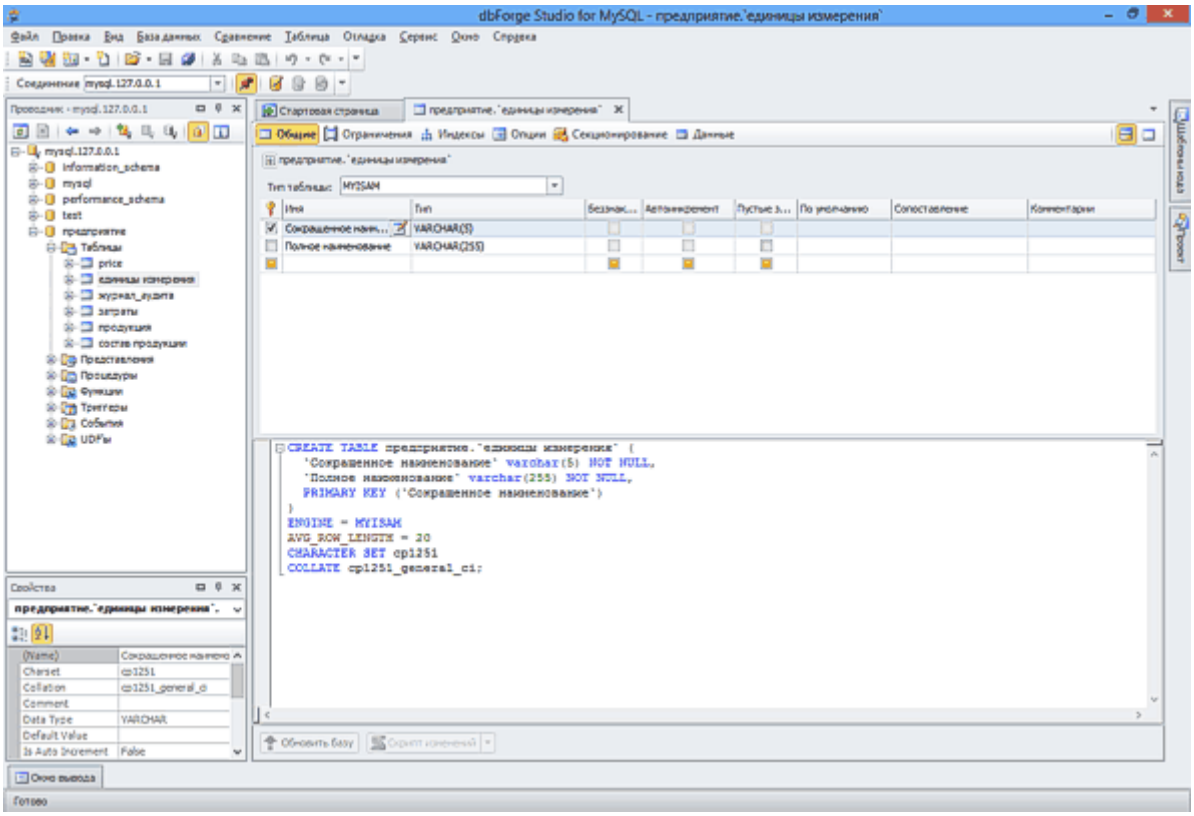
Этапы выполнения работы:

1 Запустить программу.
2 Ознакомиться с интерфейсом программы.
3 Настроить подключение к серверу MySQL.
4 Создать базу данных «Экзамен».
5 Описать этапы работы.
6 Защитить выполненную работу.

1 Интерфейс программы

Основное окно программы поделено на несколько областей (панелей):

1. Проводник баз данных. Структурированный в виде дерева перечень объектов баз данных, объединенных в функциональные группы.
2. Свойства. Отображает свойства выбранного объекта.
3. Главное окно. Рабочее окно, в котором осуществляются все действия с базой данных.
4. Проект.
5. Окно вывода. Вывод сообщений системы.

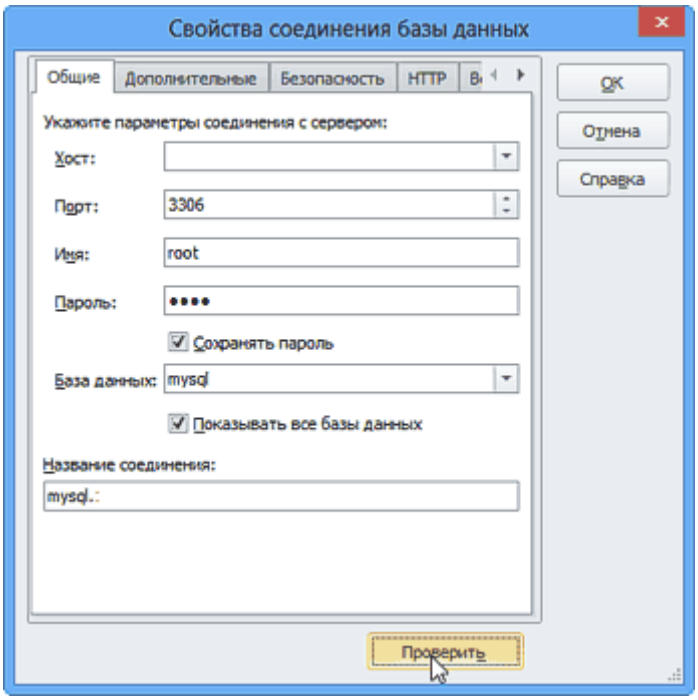


Если вы не наблюдаете произведенных изменений, то попробуйте на панели проводника программы вызвать контекстное меню на имени соединения и выбрать из него пункт "Обновить", либо нажать функциональную клавишу <F5>, либо на панели инструментов окна проводника щелкнуть кнопку  "Обновить информацию схемы".

2 Подключение к серверу mysql

После загрузки программы следует соединиться с сервером. Если это делается впервые, то на панели инструментов окна проводника необходимо щелкнуть кнопку  "Новое соединение", после чего появится окно "Свойства соединения базы данных", в котором указываются параметры соединения с сервером:

1. IP -адрес хоста, на котором расположена база данных.
2. Порт (по умолчанию "3306").
3. Имя пользователя.
4. Его пароль.
5. Можно также указать базу данных, с которой автоматически будет осуществляться работа после установки соединения.



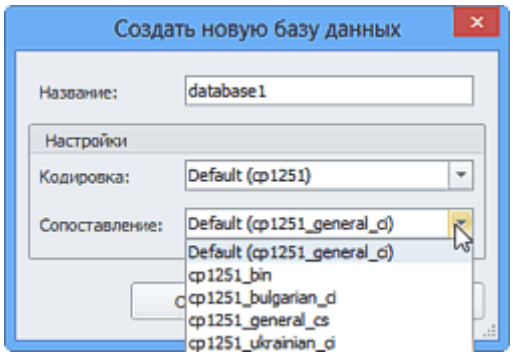
Для проверки правильности введенных параметров щелкните кнопку "Проверить". При успешном соединении с сервером последует сообщение: "Соединение установлено".

После успешного подключения к серверу на панели проводника программы появится перечень баз данных на сервере, причем на корневом уровне находится название соединения. Одновременно в окне вывода появляется сообщение об установке соединения с сервером.

При следующих загрузках dbForge Studio for SQL создавать новое соединение не нужно, просто необходимо возобновить соединение с сервером. Это можно сделать разными способами, например, вызывать контекстное меню на имени соединения на панели проводника программы и выбрать из него пункт "Открыть (соединение)" или просто дважды щелкнуть на имени соединения.

3 Создание базы данных

1. Для создания новой базы данных следует выполнить команду База данных > Новая база данных..., после чего появится окно "Создать новую базу данных".
2. В поле "Название" вводится произвольное имя.
3. В списке "Кодировка" задается кодировка данных. Естественно, при работе с кириллицей следует выбрать русскую кодировку для Windows "cp 1251".
4. В списке "Сопоставление" задается правило для работы с данными таблиц. Например, для работы с данными на русском языке выбирается набор для "cp 1251_general_ci". Вместе с тем, сопоставление позволяет "детализировать" выбранную кодировку, выбрав, например, правила для болгарского или украинского языка.



Отвлекусь немного в сторону от основной темы. Настройки (по умолчанию) в окне не случайны, они определяются при конфигурировании MySQL. Именно там, если вы планируете работать с данными на русском языке, следует задать такую установку:

character-set-server=cp1251

После создания новой базы данных в dbForge Studio for SQL ее название должно отобразиться в окне проводника.

Свойства базы данных, в частности набор символов для работы с базой данных и набор правил для работы отображаются на панели свойств при установке курсора на имени базы данных в окне проводника.

Практическое занятие № 11

Часть 1 Работа с таблицами

Этапы выполнения работы:

1. Запустить программу.
2. Настроить подключение к серверу MySQL.
3. Выбрать базу данных «Экзамен».
4. Создать в базе данных таблицы: Факультеты (id, Факультет); Студенты (id, ФИО,id_группы); Группы (id, Группа, id_факультета); Оценки (id, Оценка, Дата, id_дисциплины, id_студента); Дисциплины (id, Дисциплина).
5. Заполнить таблицы (по 15 записей).
6. Продемонстрировать созданную базу преподавателю, защитить выполненную работу.

Доступ к таблицам базы данных осуществляются в узле "Таблицы" соответствующей базы данных панели "Проводник".

1 Создание таблицы

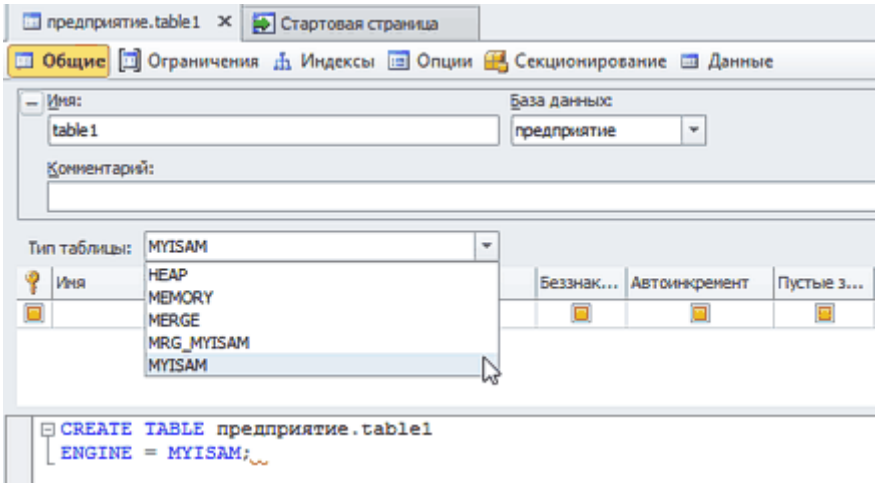
Создание новой таблицы осуществляется так:

1. Раскрываем узел с именем нужной базы данных (раскрытие и свертка узла осуществляются также, как и в "обычном" проводнике Windows, то есть щелчком на символе "+" либо "-" перед названием базы данных.

2. Вызывать контекстное меню на пункте "Таблицы" и выбрать из него пункт "Новая таблица".

3. В рабочем окне появится окно-вкладка таблицы, которая в свою очередь содержит несколько вкладок. На вкладке "Общие" задаются общие атрибуты таблицы: ее имя, тип данных, здесь же можно при необходимости изменить базу данных, для которой создается таблица. Тип данных таблицы, опять же, устанавливается автоматически соответственно настройкам MySQL.

4. На той же вкладке "Общие" вводятся данные о структуре таблицы, то есть. обо всех ее полях. Новое поле создается автоматически, его также можно добавить "явно", выполнив команду Таблица > Новый столбец. Вообще, пункт главного меню программы Таблица предназначен именно для работы с таблицей, в том числе он содержит действия, позволяющие изменять структуру таблицы: добавлять, удалять, вставлять, редактировать поля таблицы.



Ключевые свойства полей:

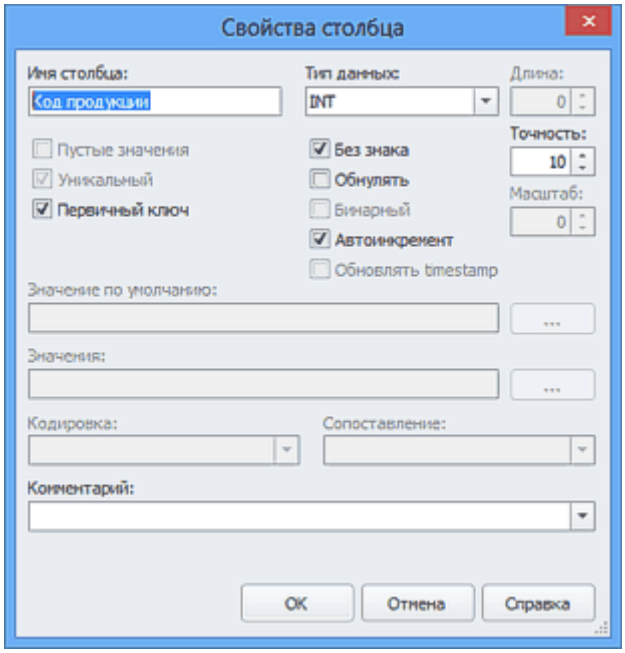
– Первичный ключ. Определение поля ключевым.

– Пустые значения. Разрешение или запрет наличия пустых значений в поле.

– Без знака. Запрет ввода отрицательных чисел в поле.

– Бинарный. Задание этого свойства определяет, что значения в этом поле будут чувствительными к регистру.

– Автоинкремент. Автоматический прирост на "1" значения в поле при добавлении новой записи. Применяется для полей целого типа данных (INT).



Для сохранения структуры таблицы следует выполнить команду Файл > Сохранить или щелкнуть соответствующую пиктограмму на панели стандартных инструментов программы.

Под структурой таблицы расположено окно "SQL", в котором отображается команда SQL, автоматически генерируемая программой во время создания таблицы по введенным нами данным. Замечу, что текст SQL-команды не редактируется.

Дальнейшее обращение к таблице осуществляется путем двойного щелчка на ее имени в проводнике, после чего ее структура отображается в главном окне.

Изменение структуры таблицы (состав полей или их свойств) осуществляется также на вкладке "Общие".

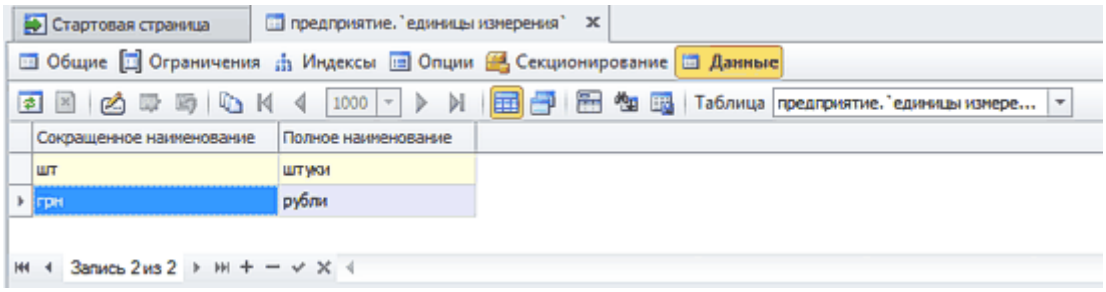
Сводные данные о таблице можно отобразить на панели свойств двойным щелчком на названии таблицы на панели проводника.

2 Ввод данных

Для ввода данных следует выбрать в проводнике нужную таблицу, а в окне-вкладке с таблицей перейти на вкладку "Данные".

Новая строка (запись) добавляется автоматически при нажатии функциональной клавиши <вниз> при установленном на последней записи курсоре. Переход от поля к полю осуществляется клавишами навигации, <Enter> или <Tab>. Если данные визуально не полностью отображаются в поле, его можно увеличить, перетянув его правую границу или дважды нажав на ней в строке с заголовками полей.

Перемещение по данным таблицы осуществляется с помощью клавиш навигации. Нижняя часть окна с таблицей содержит также панель навигации, которой удобно пользоваться при перемещении по таблице с большим количеством записей.



Отмечу, что после ввода в таблицу данных их нужно сохранить, то есть, выполнить команду Файл > Сохранить или щелкнуть соответствующую пиктограмму  на панели стандартных инструментов программы. Впрочем, если Вы и забудете об этом, то программа сама напомнит Вам о наличии несохраненных данных при окончании сеанса работы с нею.

Часть 2 Работа с запросами dbForge

Этапы выполнения работы:

1. Запустить программу.
2. Настроить подключение к серверу MySQL.
3. Выбрать базу данных «Экзамен».
4. Построить запросы по описанным ниже правилам.
5. Продемонстрировать созданные запросы преподавателю, защитить выполненную работу.

1 Построение и выполнение запроса

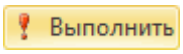
1. Выполнить команду Файл > Создать > Запрос или на панели стандартных инструментов программы щелкнуть кнопку создания нового запроса  , после чего в главном окне появится вкладка "Запрос.sql".

2. Перетянуть из окна проводника на вкладку "Запрос.sql" таблицы, из которых нужно отобрать информацию.

3. Сформировать параметры запроса, о чем речь пойдет ниже.

4. При необходимости в колонке "Псевдоним" ввести содержательное название для поля. В этом случае именно оно будет отображаться в выводе (в противном случае отображается название поля).

5. Сохранить запрос, выполнив команду Файл > Сохранить запрос.sql или щелкнув соответствующую пиктограмму на панели инструментов. Появится окно "Сохранить файл как", в котором следует указать имя запроса, которое может быть и на русском языке. Файл запроса имеет расширение SQL.

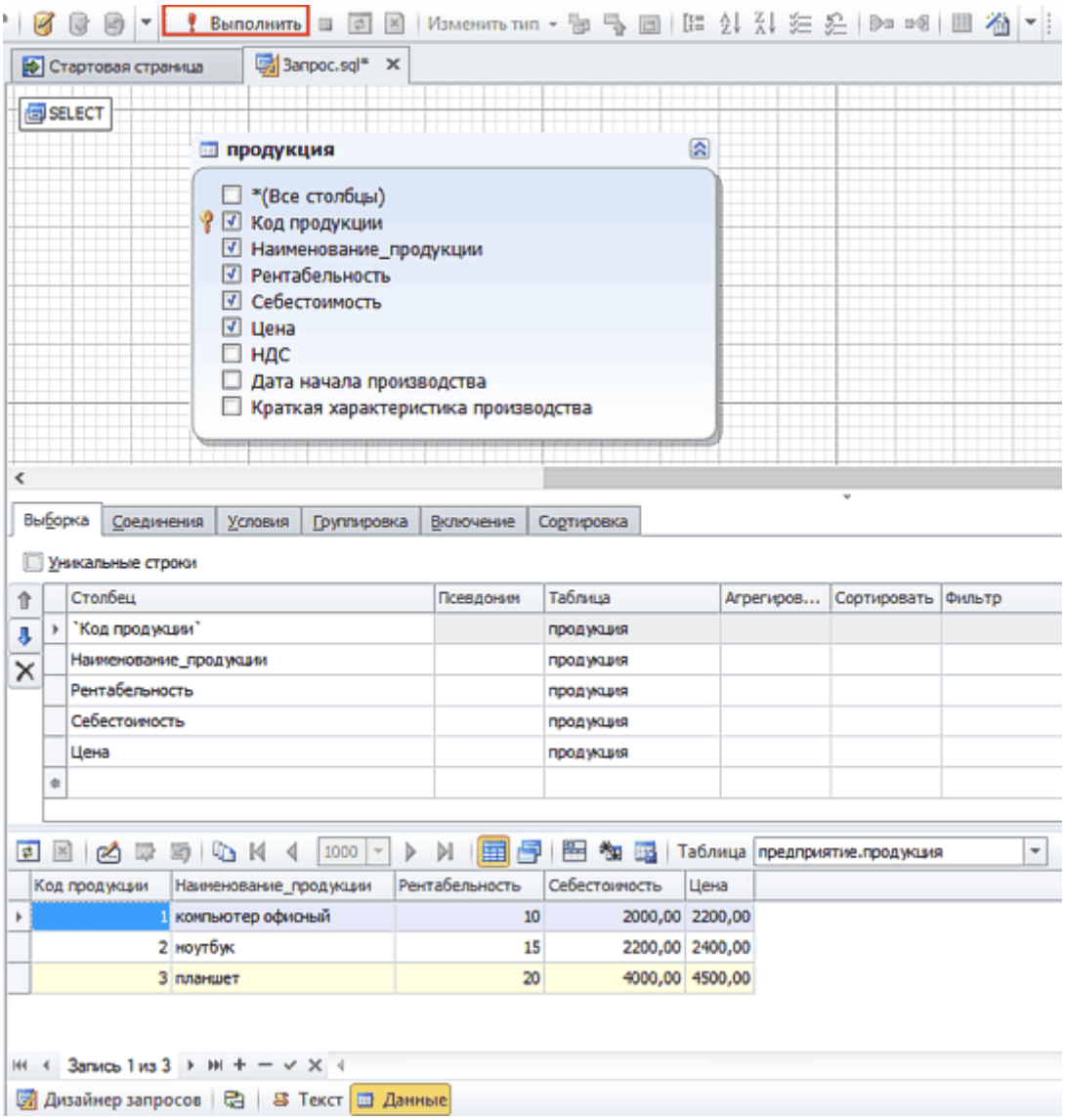
6. Для выполнения запроса следует сначала его открыть. С этой целью следует щелкнуть кнопку  "Открыть" на панели стандартных инструментов программы. Для выполнения запроса щелкните кнопку 

на панели запросов.

2 Создание запроса на выборку

Самым простым типом запроса является запрос на выборку. Отображение и формирование перечня полей, включаемых в выборку, осуществляется на вкладке "Выборка". Для включения в запрос значений поля достаточно установить флажок для поля-метки с названием поля в области окна, отображающего структуру выбранных таблиц. Другой вариант формирования запроса заключается в выборе полей из раскрывающегося списка в колонке "Столбец". Так в запрос добавляются составные выражения, содержащие, например, функции или представляющие собой выражение из нескольких полей.

Результаты запроса будут отображаться именно в том порядке, как они расположены в колонке "Столбец". Если возникает необходимость отображать определенную информацию только один раз, то нужно установить флажок для поля-метки "Уникальные строки". Для просмотра результатов запроса еще во время его формирования (без запоминания) нужно щелкнуть кнопку "Выполнить".



Задание:
1. Вывести данные о студентах.
2. Что является результатом запроса: Select Distinct ФИО From Студенты?

3 Отбор по критерию

Наложение условия на значения поля или нескольких полей осуществляется на вкладке "Условия".

Для добавления условия следует щелкнуть кнопку . Это приведет к появлению текста, который дважды содержит фразу "Введите значение", разделенную знаком "=".

1. Первый текст "Введите значение" является левой частью условия. Это - поле (название столбца) какой-то таблицы.

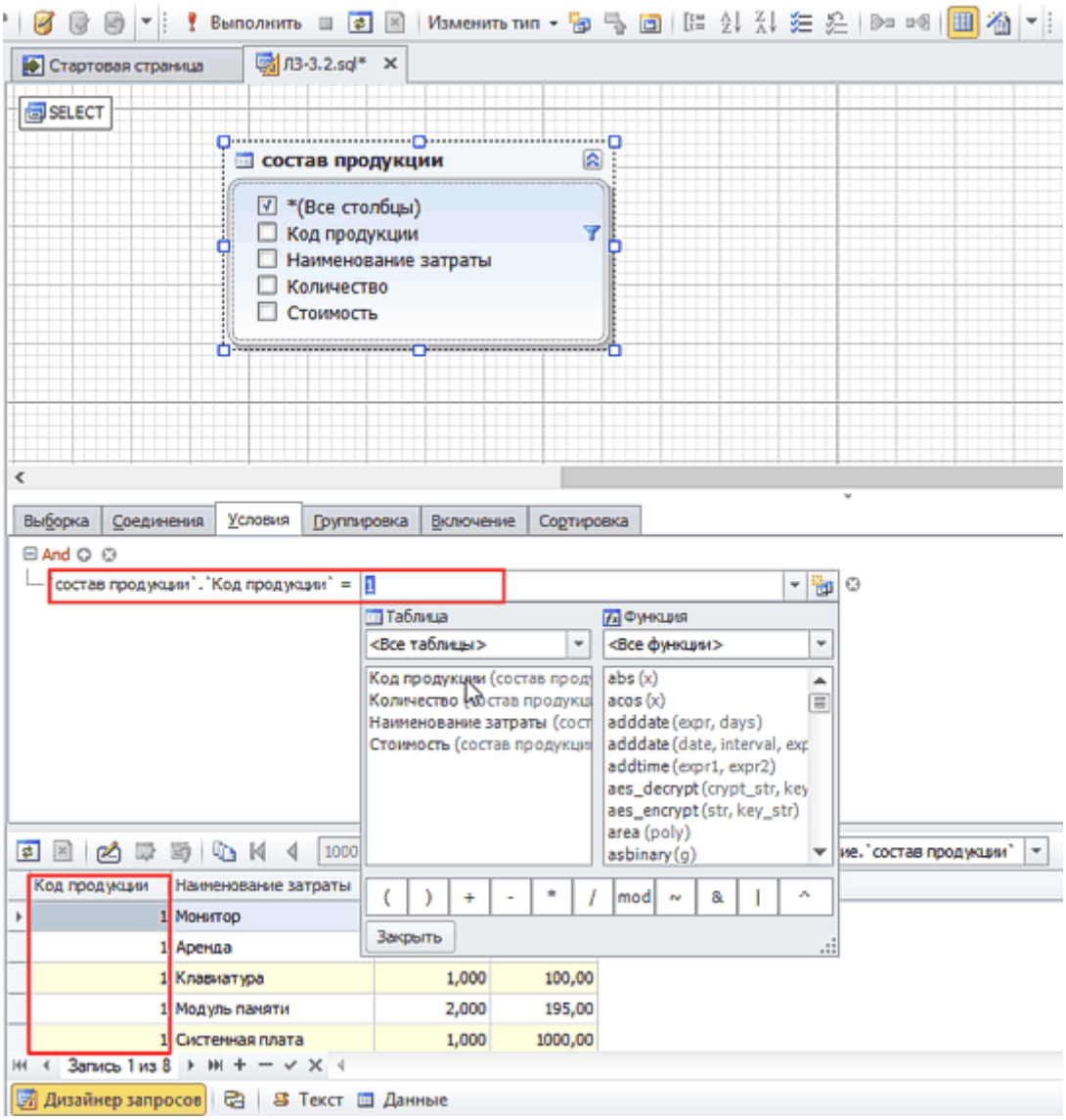
2. Знак "=" является оператором условия. Его можно заменить на другой. Для этого следует щелкнуть на нем и выбрать из появившегося списка нечто иное, например, ">".

3. Текст "Введите значение" справа от знака "=" является значением условия. Щелчок на тексте "Введите значение" приводит к отображению списка полей таблиц, различных выражений и пр., что можно представить как условие.

Например, для отбора студентов, имеющих оценку "5", условие может быть записано как "Оценка"=5.

Условий может быть несколько. Для добавления нового условия также необходимо щелкнуть кнопку [+] и сформировать по вышеприведенным правилам новое условие.

По умолчанию между собой условия соединяются логическим оператором AND, однако и его можно изменить, щелкнув на нем и выбрав из списка другой.



Задание:

1. Вывести данные о студентах для id>3.
2. Вывести данные из таблицы Оценки для условия: Дата = 22.12.2018, Дата = 24.12.2018, Дата = 26.12.2018. Выполнить запрос двумя способами: 1) логические операторы; 2) оператор in.
3. Вывести информацию о тех, кто сдал сессию.
4. Вывести информацию о студентах, фамилии которых начинаются на П.
5. Что является результатом запроса: Select ФИО From Студенты Where ФИО Like ‘Р%ин _._.’?

4 Создание связей (объединение таблиц)

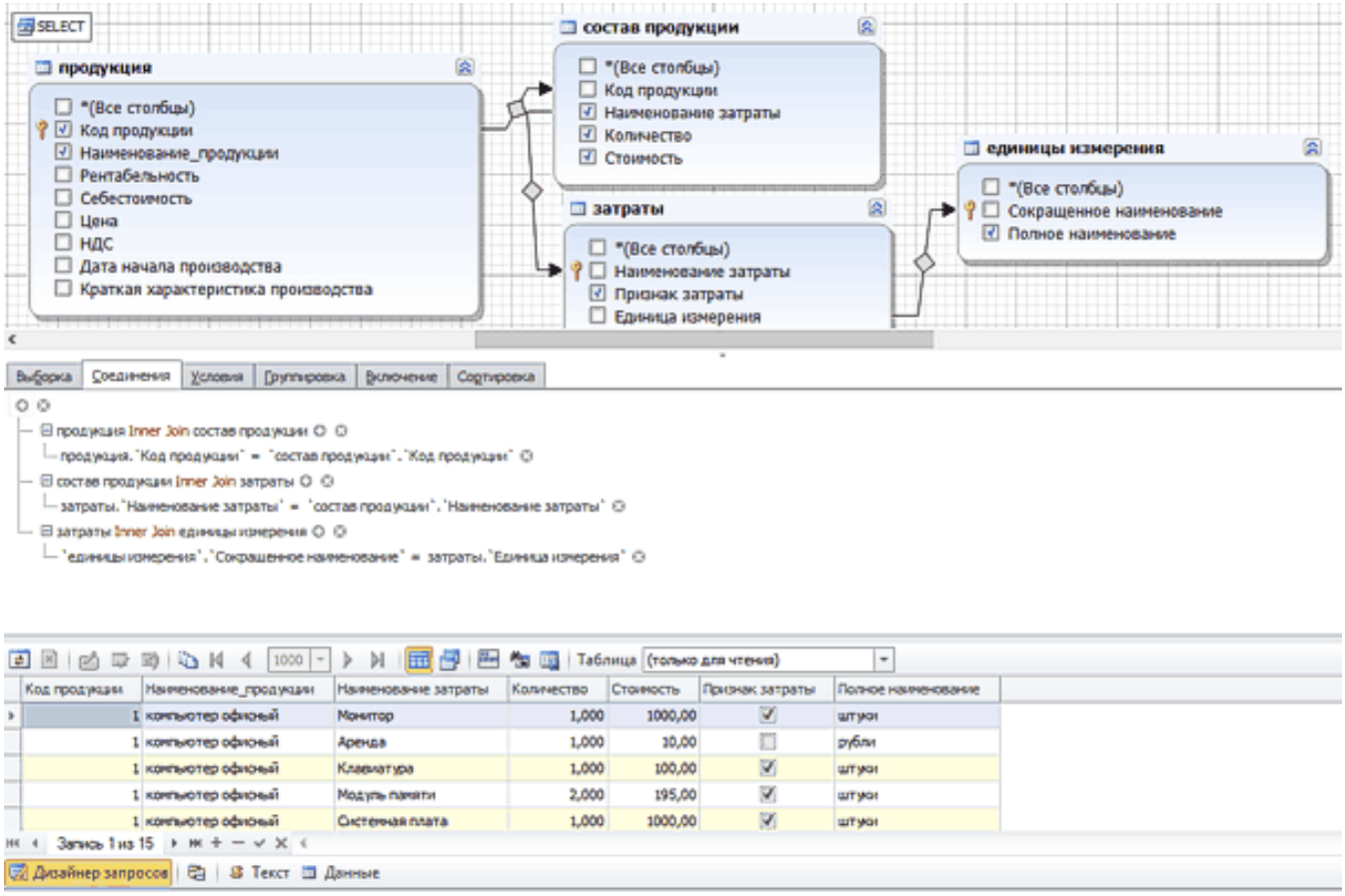
В базе данных MySQL нельзя одноразово жестко зафиксировать связи между таблицами. Пользователь в каждом запросе должен создавать их с помощью SELECT и JOIN. Программа позволяет создавать пять видов связей, в том числе внутреннюю, левостороннее и правостороннее объединение.

Объединение двух таблиц осуществляется путем объединения мышкой нужных полей таблиц. Созданная связь отображается графически. Те же действия можно произвести и на вкладке "Соединения". Для этого нужно щелкнуть кнопку , после чего появится пустой оператор JOIN. Щелкаем на тексте "Укажите имя столбца" и добавляем к соответствующим полям поля объединяемых таблиц.

Код команды SELECT, отображающий создание связей, также отображается на вкладке "Соединения".

Для изменения типа связи следует вызывать контекстное меню на типе (названии) соединения, например, "Inner Join" и выбрать из него нужный.

Удалить связь можно разными способами, например, вызывав контекстное меню на графической линии связи и выбрав из него пункт "Удалить из диаграммы".



Задание:

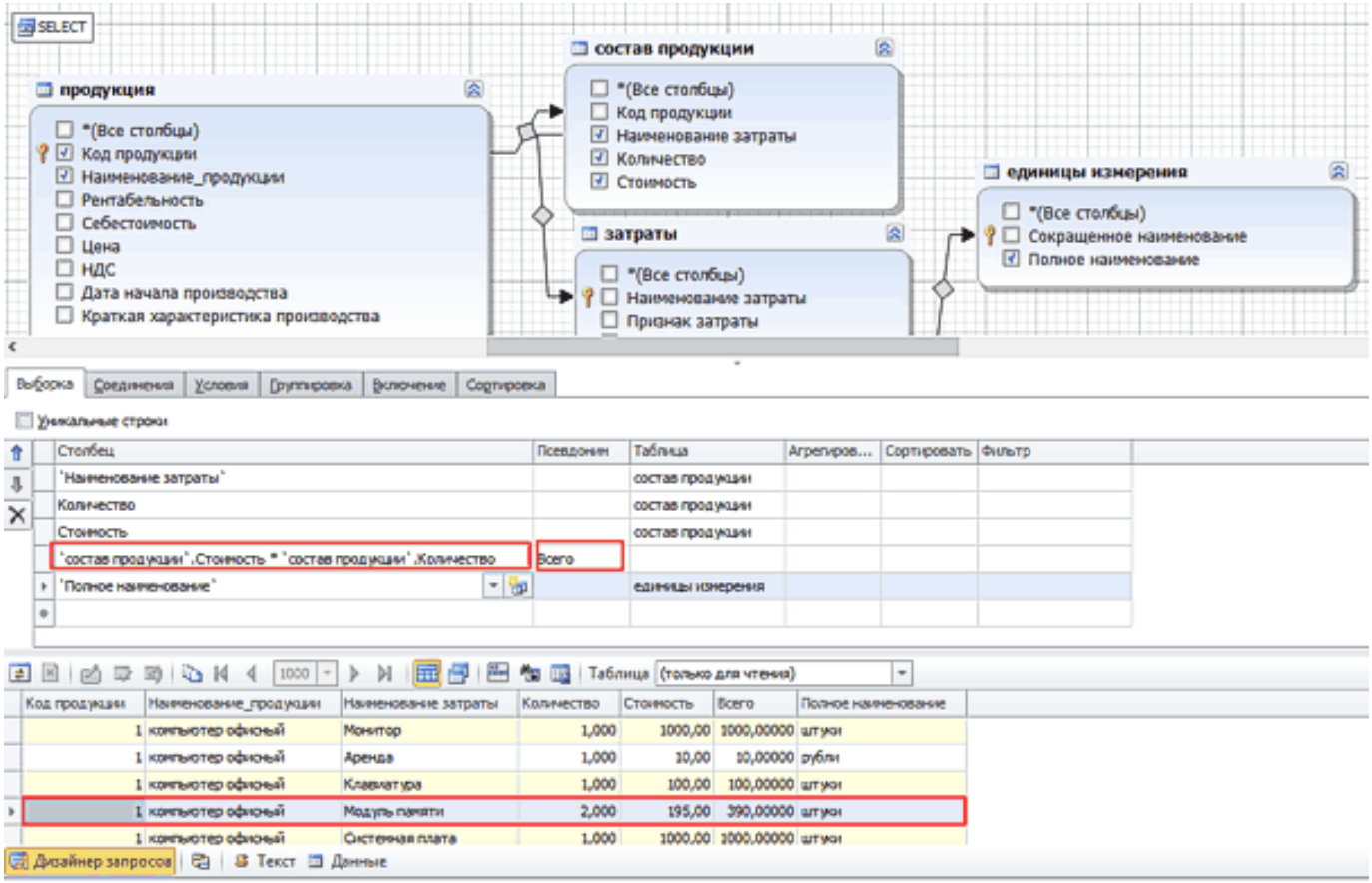
1. Вывести оценки каждого студента.
2. Вывести группы с указанием факультета.

5 Вычисляемые поля

Достаточно часто вывод результатов запроса должен содержать не просто значения одного из полей таблицы, а комбинацию значений полей из одной или нескольких таблиц или математическую операцию над ними. Например, в запросе следует подсчитать общую стоимость товара, которая определяется как произведение общего количества товара и цены за единицу товара. Для этого следует создать вычислительное поле. Например, создание произведения полей "Цена" и "Количество" осуществляется так.

1. Перейти на вкладку "Выборка".
2. Щелкнуть на свободной строке в поле "Столбец".

3. Выбрать поле "Цена".
4. В нижней части окна выбора щелкнуть пиктограмму оператора "*".
5. Из соответствующей таблицы выбрать поле "Количество", после чего в поле "Столбец" появится выражение Цена*Количество.
6. Для создания содержательной подписи для вычислительного поля следует в поле "Псевдоним" заменить выражение "Exp" на "Всего".



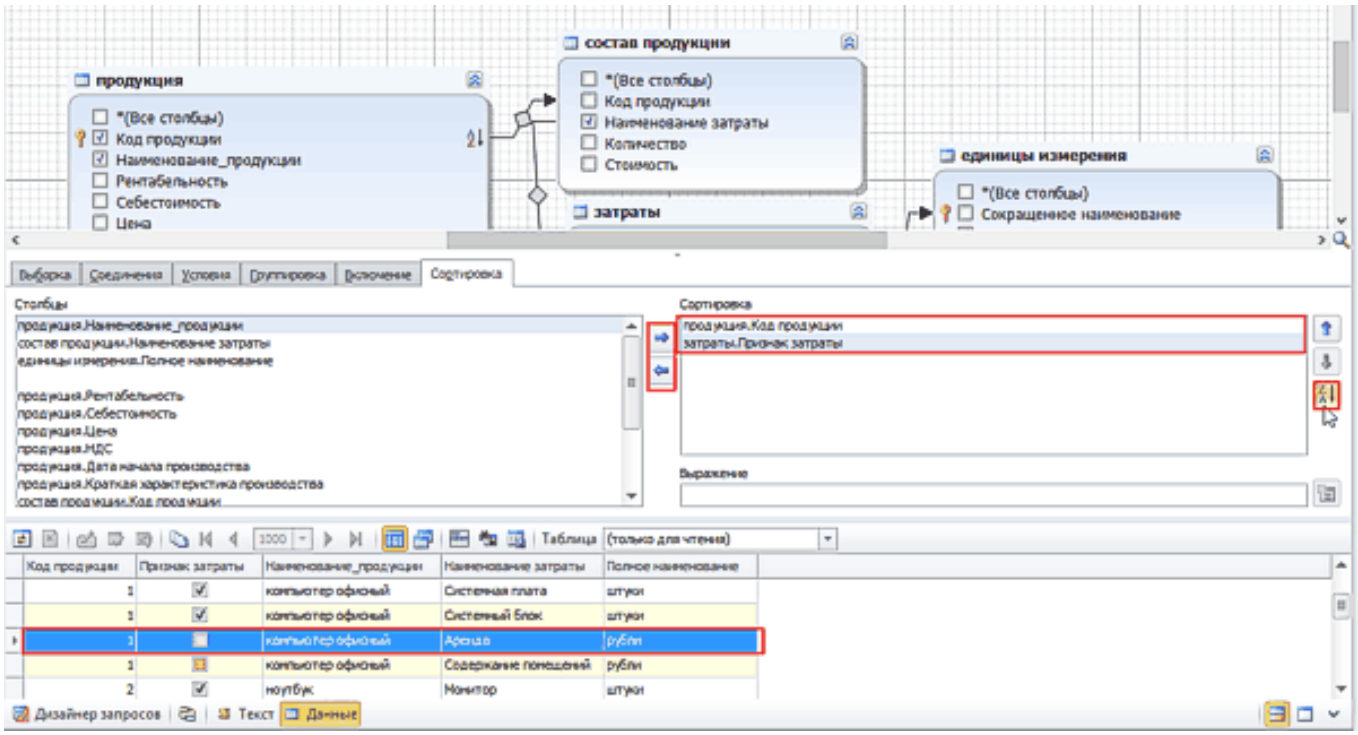
Задание:

1. Что является результатом запроса: Select ‘Группа-’+Upper(Группа) As ‘Номер группы’ From Группы?

6 Упорядочивание (сортировка) записей

Если необходимо упорядочить отобранные данные по определенному полю, то применяют, как известно, сортировку. Определение полей сортировки осуществляется на вкладке "Сортировка". Здесь список "Столбцы" содержит перечень всех полей, включенных в запрос. Двойной щелчок на любом из них или щелчок кнопки "Добавить столбец"  приводит к переносу этого поля в список "Сортировка", который и определяет сортировочные поля и порядок упорядочивания записей. По умолчанию применяется метод сортировки "по возрастанию", но его очень просто изменить на "по убыванию", щелкнув кнопку "Сортировать по..."  .

Количество полей сортировки не ограничивается, что позволяет осуществить многоуровневую сортировку результатов вывода, упорядочив его сразу по нескольким полями. В этом случае поля, для которых осуществляется сортировка, следует расположить в списке в именно том порядке, в котором она будет осуществляться.



Задание:

1. Вывести данные о студентах и их оценках, упорядочив фамилии по возрастанию.
2. Вывести данные о группах и студентах, упорядочив по возрастанию каждое поле.
6. Что будет являться результатом запроса: Select id_группы, ФИО From Студенты Where id>10 Order By 1 Asc, 2 Desc?

7 Группировка записей и агрегатные функции

Действие группировки дает возможность объединить одинаковые по какому-то признаку записи таблицы для определенного поля в группы и применять к ним вычисления с помощью разных функций.

Группировка записей задается на вкладке "Группировка".

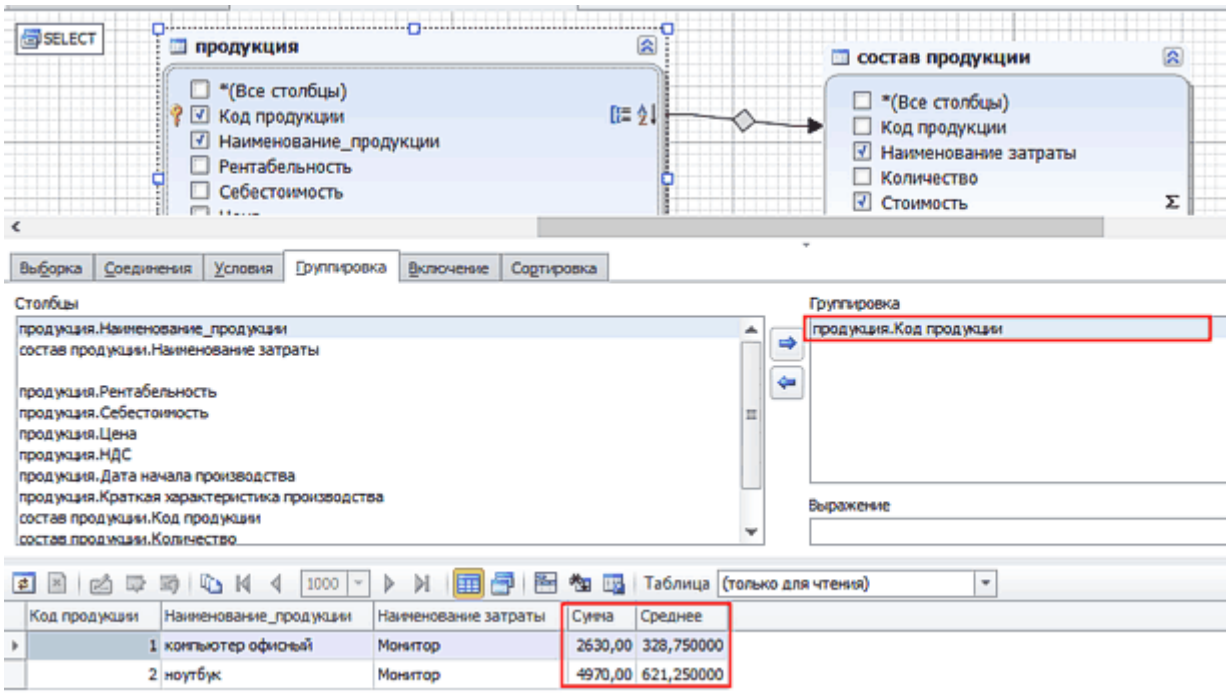
Добавление функции для поля осуществляется на вкладке "Выборка" так:

1. Щелкнуть кнопку нового поля в колонке "Столбец".
2. В списке "Функция" выбрать нужную функцию, после чего ее имя появится в колонке "Столбец".
3. В качестве аргумента функции выбирается нужное поле из списка "Таблица".

Напомню, что к агрегатным функциям относятся следующие:

- функция Sum предоставляет возможность подсчитать для группы записей сумму;
- функция Avg - вычислить среднее значение;
- Count - подсчитать общее количество записей;
- Max и Min - определить максимальное и минимальное значение в поле.

Так, если осуществить группировку студентов по их фамилии, номеру зачетной книжки и т.п., то в таблице, которая содержит данные с оценками студентов с помощью функции Sum можно подсчитать общую сумму оценок для каждого студента, а с помощью функции Avg - вычислить его средний балл.



Задание:

2. Вывести средний балл по группам.
3. Вывести количество студентов по каждой дисциплине, не имеющих оценки 3.

8 Использование агрегатных функций в условиях

В условиях во фразе HAVING можно использовать агрегатные функции, которые действуют в пределах создаваемых групп. Это осуществляется на вкладке "Включение".

Например, для отбора оценок студентов группы, которые имеют оценки выше среднего балла, можно применить два варианта:

Оценка > AVG(Оценка)

Оценка - AVG(Оценка) > 0

Одновременно на вкладке "Группировка" должна быть задана группировка по студентам.

Задание:

1. Вывести количество отличников.
2. Вывести средний балл по информатике

Часть 3 Формирование триггеров

Этапы выполнения работы:

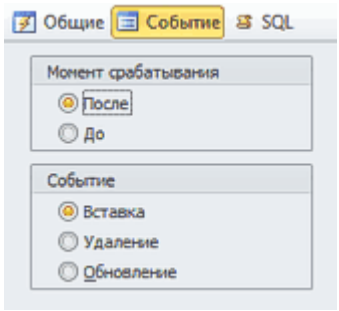
1. Запустить программу.
2. Настроить подключение к серверу MySQL.
3. Выбрать базу данных «Экзамен».
4. Построить триггеры.
5. Продемонстрировать созданные команды преподавателю, защитить выполненную работу.

Как известно, **триггер** - это процедура (последовательность SQL-операторов), которая активируется во время выполнения операций манипулирования данными (добавление, замена и удаление) над таблицами. Триггеры являются одним из механизмов поддержки целостности базы данных.

Программа dbForge Studio for SQL представляет автоматизированные средства создания триггеров.

Создания триггера осуществляется по такому алгоритму:

1. На панели проводника вызывать контекстное меню для пункта "Триггеры" и выбрать из него пункт "Новый триггер". Появится окно формирования триггера.
2. На вкладке "Общие" определяются общие реквизиты триггера: имя, владелец и таблица, в качестве которых выступают соответственно база данных и таблица, для которых создается триггер.
3. В поле "Тело триггера" на той же вкладке разработчиком самостоятельно формируется последовательность операторов, которые будут выполняться во время выполнения операции манипулирования данными. Они располагаются между операторами BEGIN и END.
4. На вкладке "Событие" определяется, для какой операции манипулирования данными предназначен триггер, а также момент его срабатывания: до или после события.



На вкладке "SQL" отображается автоматически сформированный системой скрипт для триггера.

```
CREATE
  DEFINER = 'root'@'127.0.0.1'
TRIGGER предприятие.`Добавление записи в таблицу продукции`
  AFTER INSERT
  ON предприятие.продукция
  FOR EACH ROW
BEGIN
  INSERT INTO журнал_аудита SET Пользователь = USER(), Операция = 'Вставка';
END
```

Часть 4 Резервное копирование / восстановление

Этапы выполнения работы:

1. Запустить программу.
2. Настроить подключение к серверу MySQL.
3. Выбрать базу данных «Экзамен».
4. Выполнить резервное копирование.
5. Провести восстановление базы данных.
6. Описать выполненные действия, защитить выполненную работу.

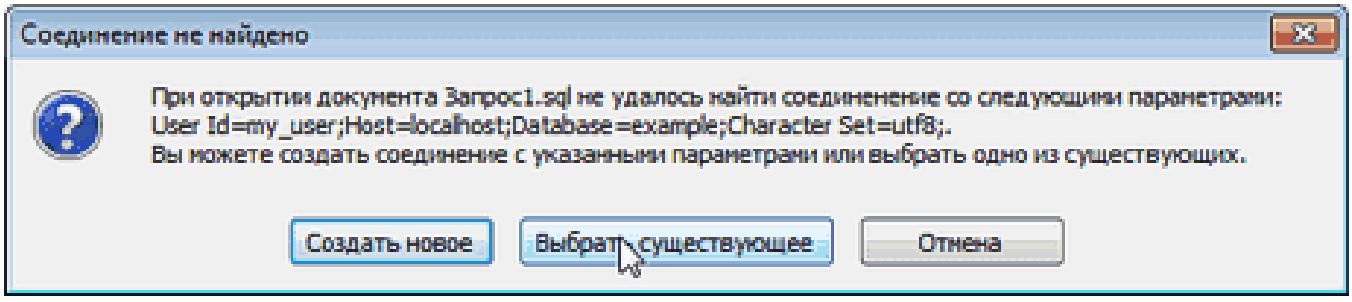
При работе с компьютером всегда необходимо помнить о возможности аварийных ситуаций: отключение электричества, выхода из строя устройств и т.п. Если база данных содержит значительный объем информации, то потеря или разрушение данных может быть воспринято как "конец света". Поэтому первое правило при работе с любой системой - это наличие резервной копии данных, что позволяет при необходимости восстановить данные. Копия также может быть полезной и при необходимости возврата к предыдущим данным.

Создание резервной копии осуществляется командой База данных > Резервная копия > Создать резервную копию БД. Архив создается с расширением SQL, при этом его имя содержит имя базы данных, для которой он создается и время его создания, например: "предприятие 20101117 1559".

Для восстановления базы данных желательно ее наличие на сервере. Впрочем, ее можно создать и автоматически в процессе восстановления. После этого следует вызывать контекстное меню на названии базы данных и последовательно выбрать из него пункты Резервная копия, Восстановить БД из резервной копии.

Действия по созданию и возобновлению резервной копии осуществляются под управлением программы-мастера и сложностей не вызывают. Единственно, что при создании резервной копии обращайте внимание на то, чтобы в поле "База данных" была выбрана нужная база данных.

И еще одно замечание. Следует иметь в виду, что если база данных переносится с компьютера на компьютер, то такие объекты, как таблицы со всем их содержимым и триггеры будут восстановлены корректно и без проблем. А вот для каждого запроса при первом его запуске на другом компьютере нужно будет давать подтверждение про перенос его на новый сервер.



Практическое занятие № 12-22 Подготовка курсовой работы

Работа по графику выполнения курсового проекта.

Дата	**Наименование вида работы**	**Отметка о выполнении**
10.03	Изучение Методических рекомендаций по выполнению и защите курсовой работы.	
10.03	Выбор и утверждение темы курсовой работы. Составление развернутого плана работы. Подбор литературных источников	
17.03	Написание введения.	
17.03	Написание первого параграфа первой главы. Представление промежуточных результатов научному руководителю.	
24.03	Написание второго параграфа первой главы. Работа над первой главой. Представление промежуточных результатов научному руководителю.	
24.03	Оформление подготовленной части работы	
3.04	Написание первого параграфа второй главы. Представление научному руководителю подготовленного варианта работы.	
21.04	Написание второго параграфа второй главы. Работа над второй главой. Представление промежуточных результатов научному руководителю.	
16.04	Представление научному руководителю окончательного варианта работы	
17.04	Доработка, устранение замечаний научного руководителя	
23.04	Оформление работы.	
24.04	Представление готовой работы научному руководителю	
30.04	Подготовка к защите	
07.05	Защита курсовой работы	
08.05	Защита курсовой работы	

Практическое занятие № 23-25 Разработка информационной системы средствами Visual Studio и MySql

Этапы выполнения работы:

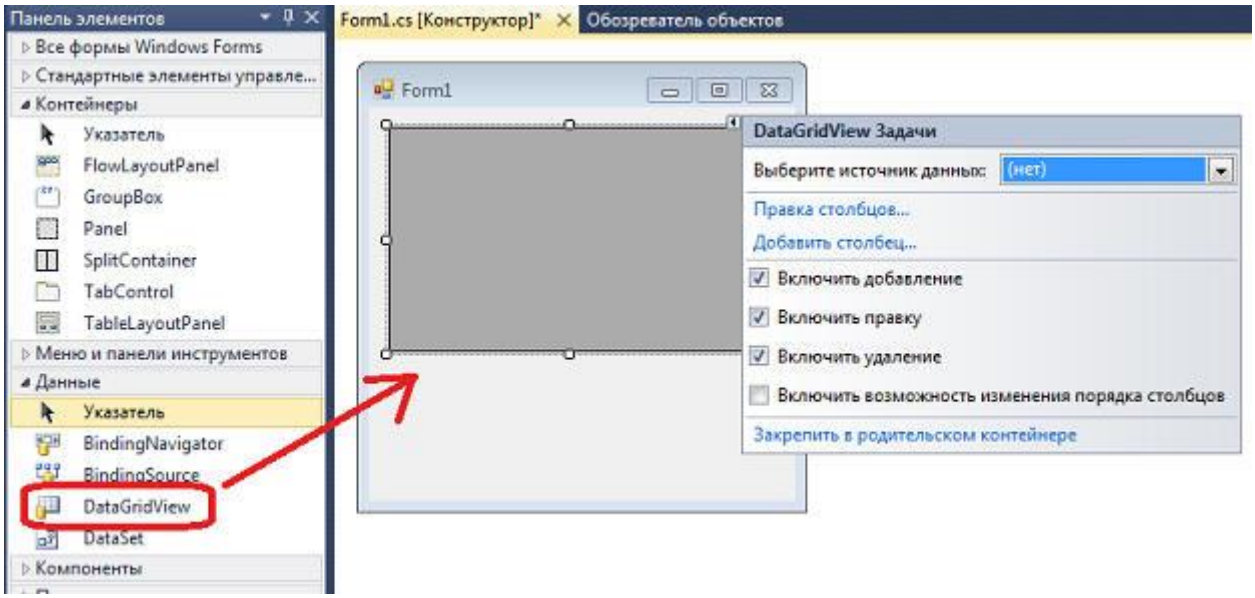
1. Запустить программу Visual Studio.
2. Выполнить действия в соответствии с описанным ниже алгоритмом.
3. Добавить две таблицы в указанную базу данных.
4. Добавить кнопки, позволяющие вносить данные, сохранять их, редактировать, производить выборку, удалять, осуществлять поиск.
5. Разработать интерфейс по теме курсового проекта.
6. Добавить кнопки, позволяющие вносить данные, сохранять их, редактировать, производить выборку, удалять, осуществлять поиск.

Алгоритм создания информационной системы

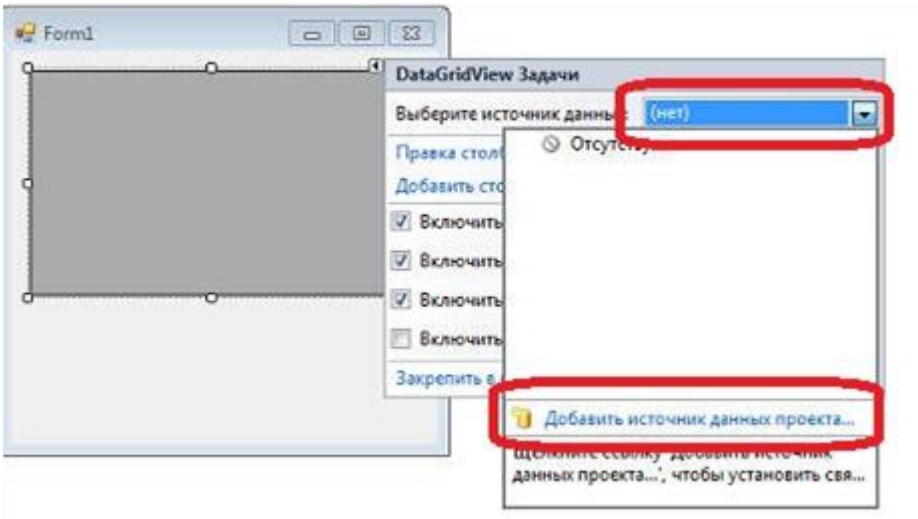
1. **Подключение базы данных SQL к проекту Visual Studio.**

На прошлых занятиях мы создали базу данных SQL – «Экзамен». Подключим эту базу данных к программе на C# через Visual Studio 2010.

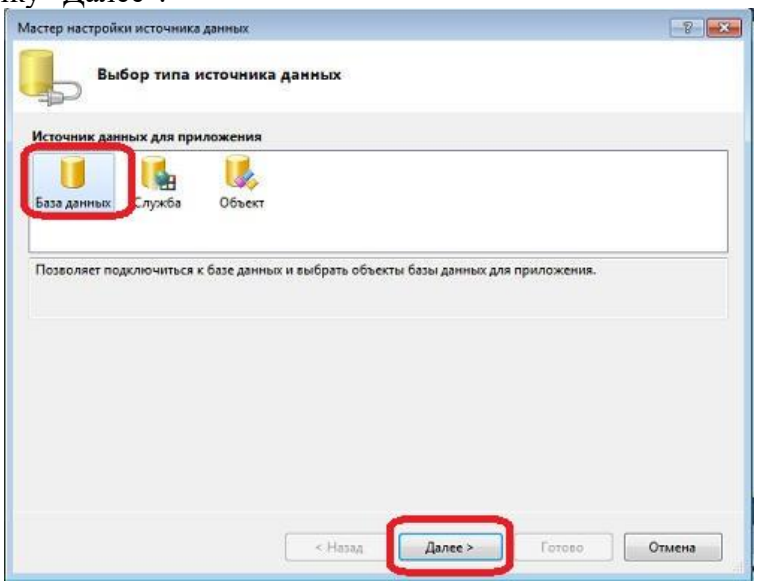
Добавляем компонент DataGridView:

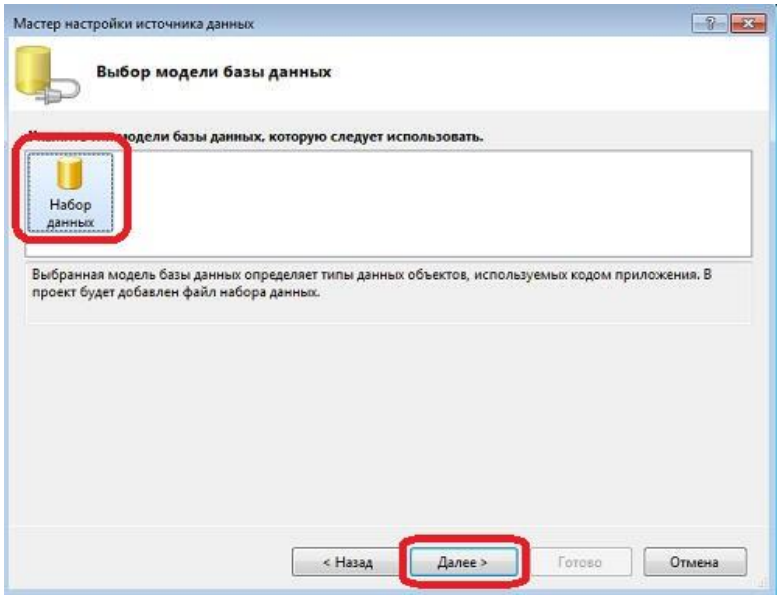


Затем надо выбрать источник данных, в данном случае придется создать новый источник данных:



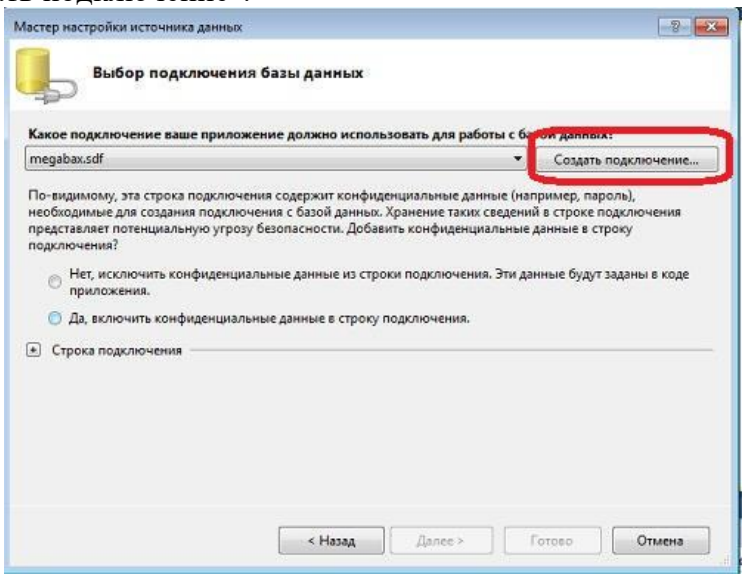
Откроется мастер настройки источника данных. Выберем "База данных" и нажмем кнопочку "Далее":



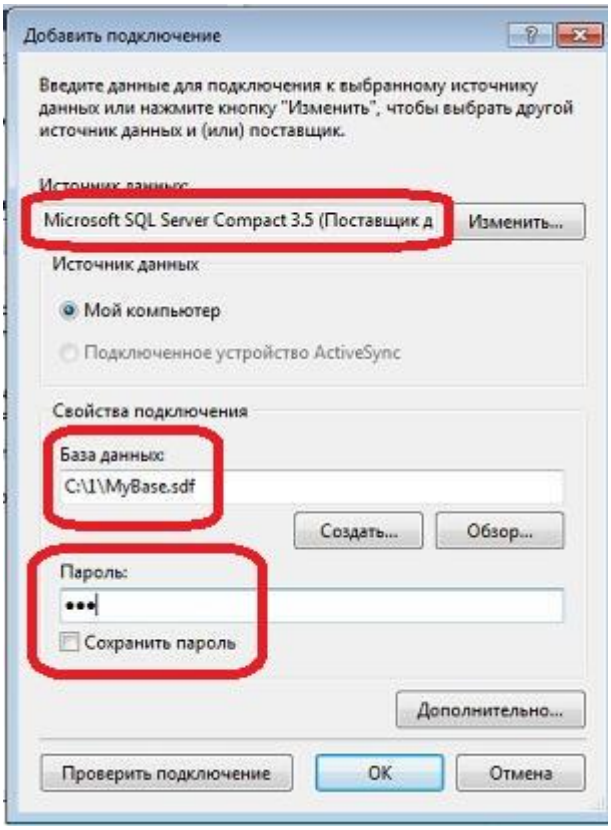


Жмем кнопочку "Далее":

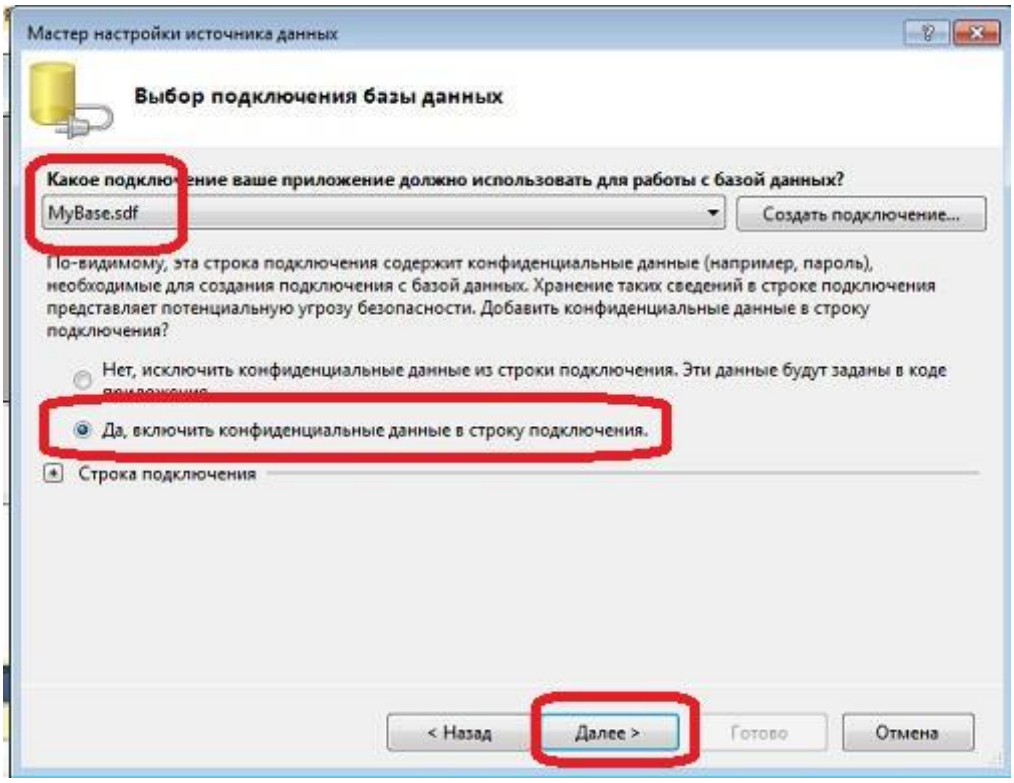
Далее, идет диалог выбора подключения соединения с базой данных. выбираем кнопку "Создать подключение":



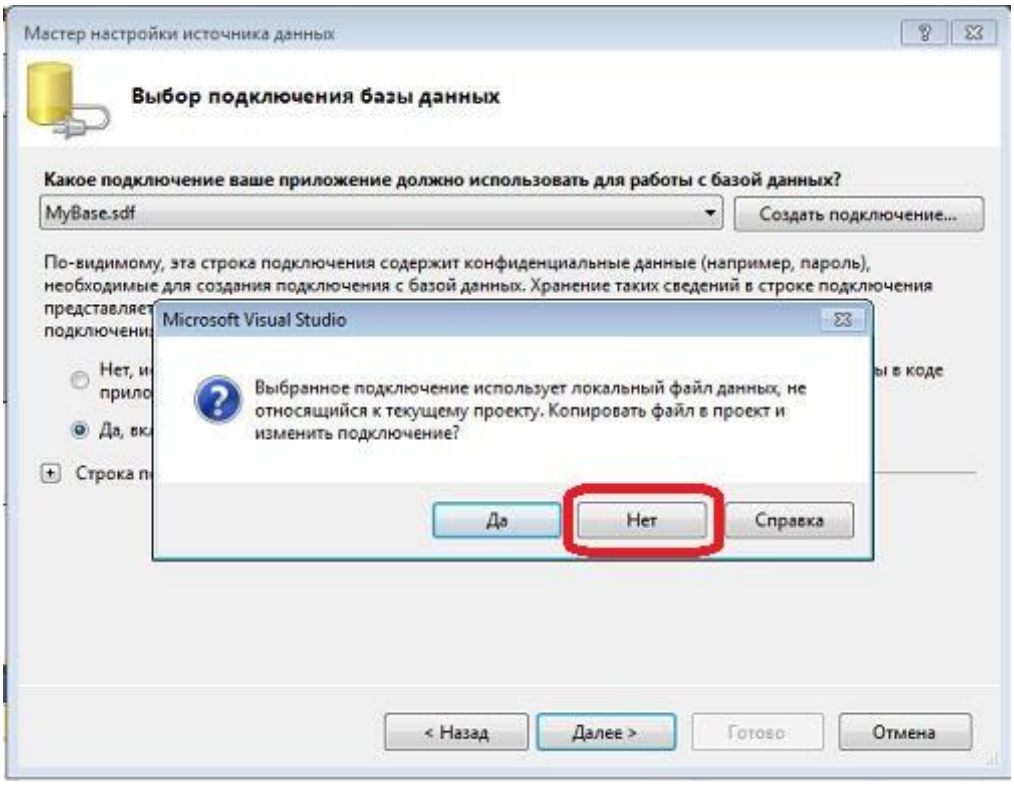
Откроется диалог создания подключения, в нем выберем источник данных "Microsoft SQL Server Compact 3.5", укажем путь к созданной базе данных и пароль к ней:



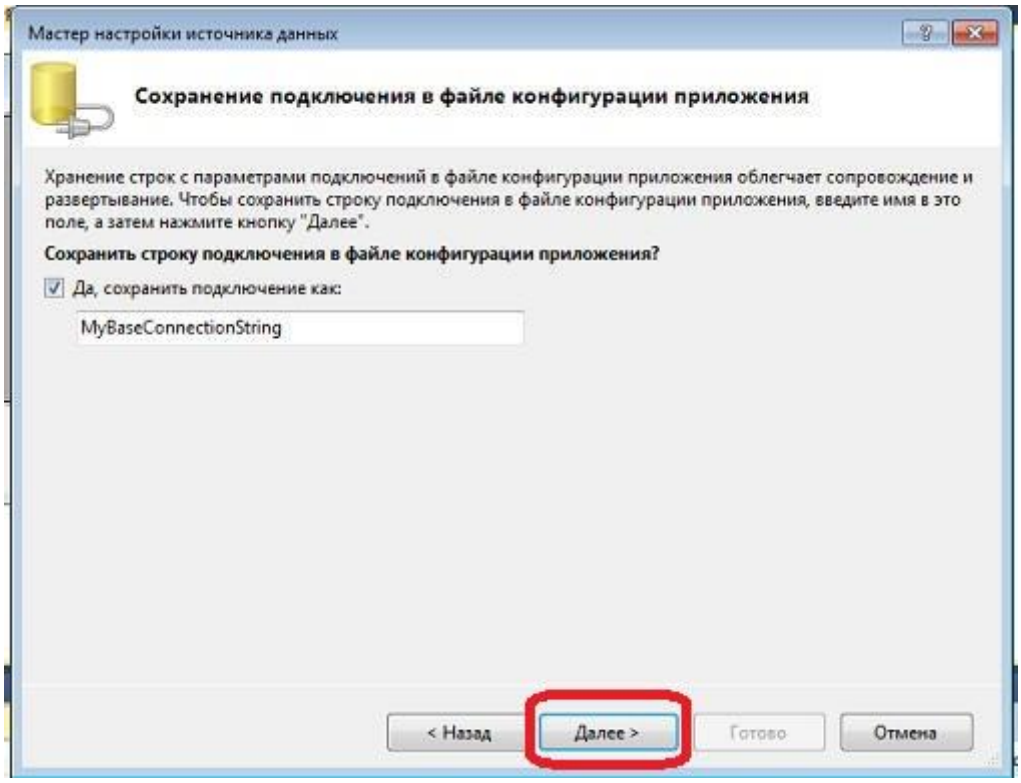
И так мы создали подключение к базе данных, теперь она у нас появилась в списке выбора. Это пока тестовый пример, поэтому включим конфиденциальную информацию в строку подключения, чтобы нам не набирать каждый раз пароль, когда будем соединяться с базой данных, и нажмем кнопочку "Далее":



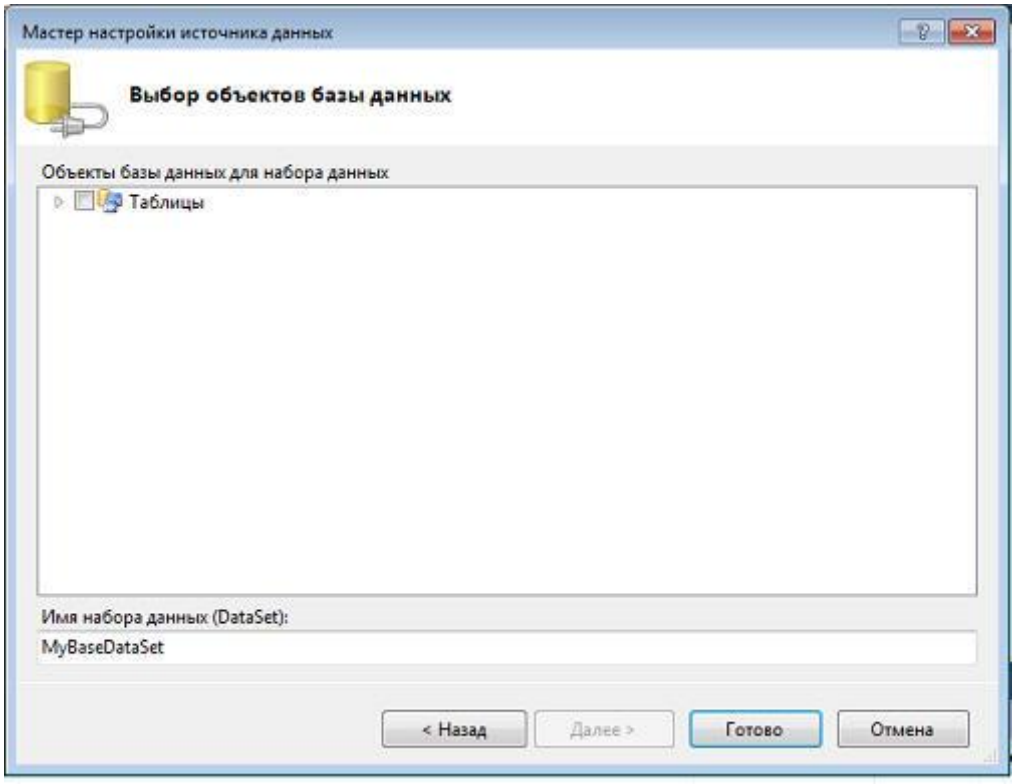
На вопрос, копировать ли базу данных в проект, ответим "Нет":



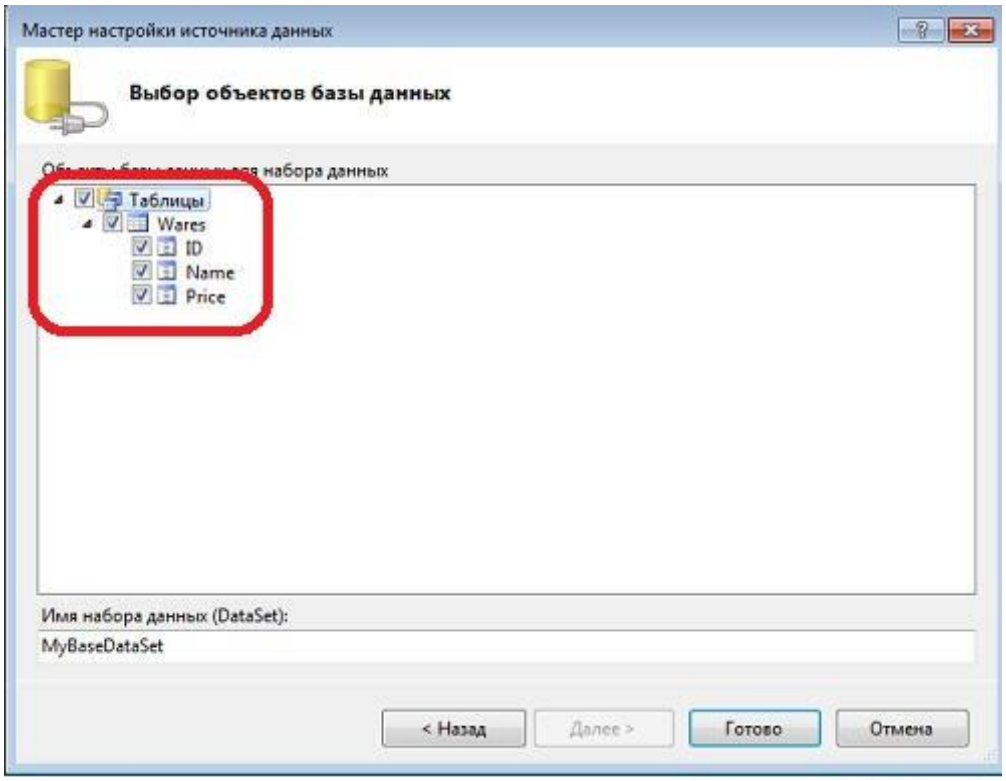
В следующем диалоге жмем "Далее":



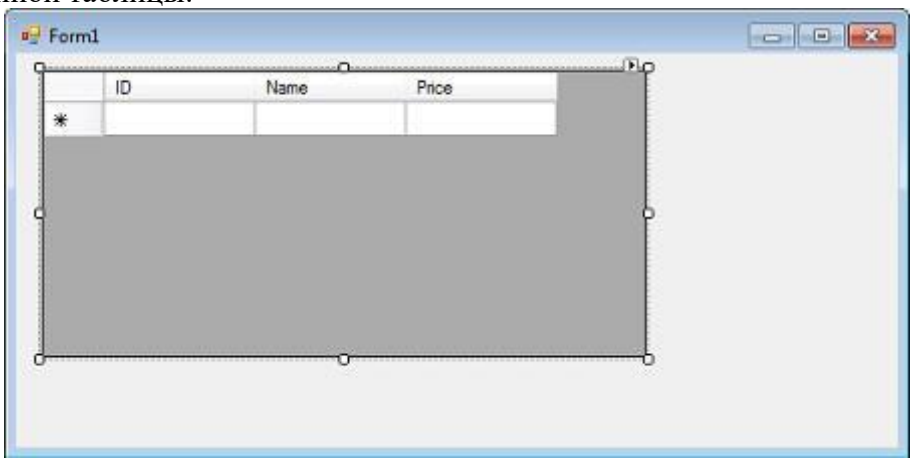
После чего у нас появиться список данных, доступных для выбора:



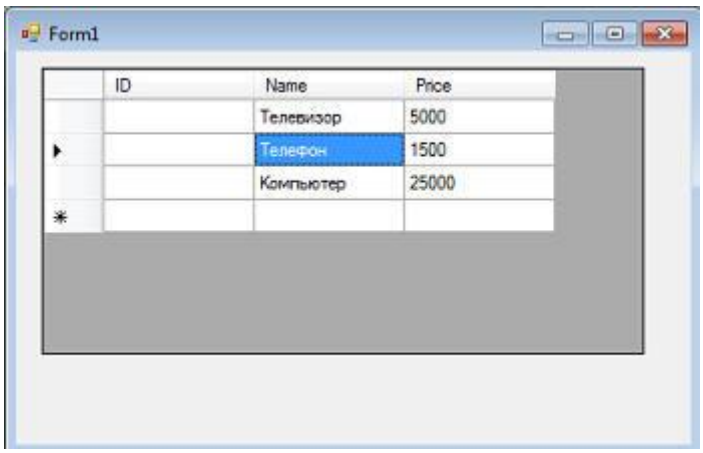
Выберем созданную нами таблицу:



После нажатия кнопочки "Готово" у нас в компоненте DataGridView отобразятся столбцы выбранной таблицы:

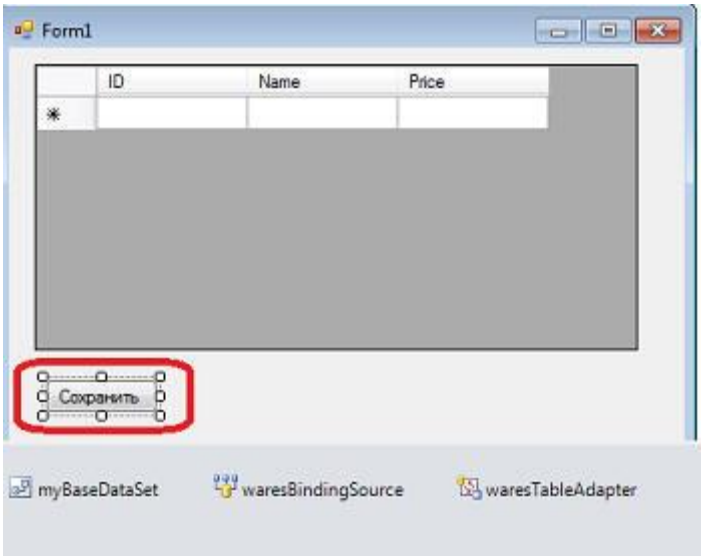


Все, теперь мы можем запустить программу и даже редактировать данные в таблице базы данных:



2. Сохранение данных

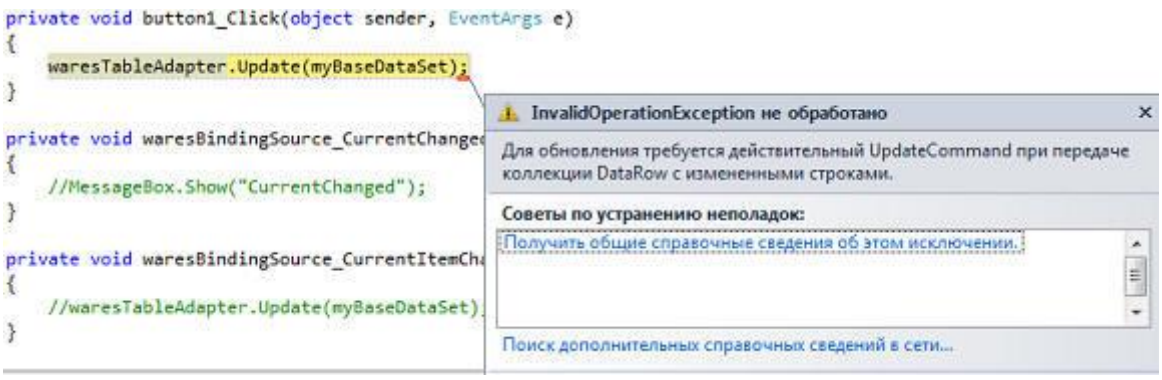
Мы подключили к проекту на Visual Studio 2010 базу данных SQL. Добавим к проекту, созданному на прошлом занятии, кнопочку "Сохранить":



В обработчике нажатия на эту кнопку у нас будет всего лишь одна строка:

```
private void button1_Click(object sender, EventArgs e)
{
   waresTableAdapter.Update(myBaseDataSet);
}
```

Примечание: кнопка "Сохранить работает" только если мы добавляем новые строки. А вот стоит только добавить новую строку, у нас программа выдает сообщение "Для обновления требуется действительный UpdateCommand при передаче коллекции DataRow с измененными строками":

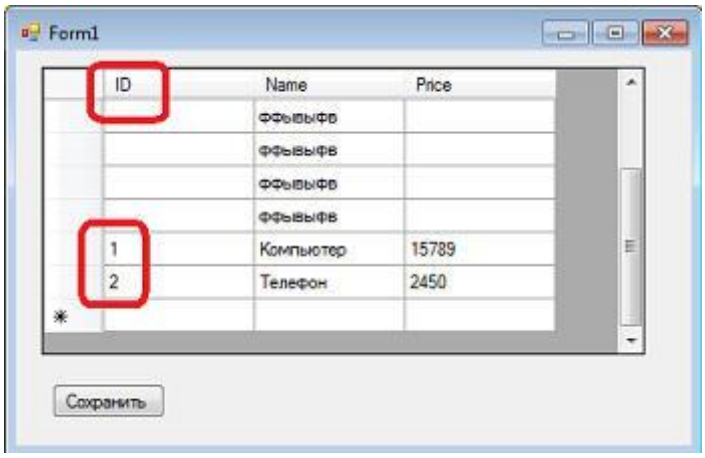


Чтобы исправить этот недочет, необходимо задать у адаптера данных UpdateCommand. Вот пример кода:

```
private void button1_Click(object sender, EventArgs e)
   {
      SqlCeCommand command = new SqlCeCommand(
```

```
            "UPDATE wares SET ID = @ID, Name = @Name, Price=@Price "+
            "WHERE (ID = @ID)");
        command.Connection = waresTableAdapter.Connection;
        waresTableAdapter.Adapter.UpdateCommand = command;

        SqlCeParameter parametr;

        //Параметр ID
        parametr = new SqlCeParameter("@ID", SqlDbType.Int);
        parametr.SourceColumn = "ID";
        command.Parameters.Add(parametr);

        //Параметр Name
        parametr = new SqlCeParameter("@Name", DbType.String);
        parametr.SourceColumn = "Name";
        command.Parameters.Add(parametr);

        //Параметр Price
        parametr = new SqlCeParameter("@Price", SqlDbType.Float);
        parametr.SourceColumn = "Price";
        command.Parameters.Add(parametr);

        waresTableAdapter.Update(myBaseDataSet.Wares);
    }
```

Теперь можно корректно редактировать наименование товара и цену. Единственно, нельзя редактировать поле ID и должны вводить его вручную при добавлении новой строки, к тому же, ID должен быть уникальным:



3. Вставка, удаление, обновление записей в базе данных

Метод **ExecuteReader()** извлекает объект чтения данных, который позволяет просматривать результаты SQL-оператора Select с помощью потока информации, доступного только для чтения в прямом направлении. Однако если требуется выполнить операторы SQL, модифицирующие таблицу данных, то нужен вызов метода ExecuteNonQuery() данного объекта команды. Этот единый метод предназначен для выполнения вставок, изменений и удалений, в зависимости от формата текста команды.

Понятие не запросный (nonquery) означает оператор SQL, который не возвращает результирующий набор. Следовательно, операторы Select представляют собой запросы, а

операторы Insert, Update и Delete — нет. Соответственно, метод ExecuteNonQuery() возвращает значение int, содержащее количество строк, на которые повлияли эти операторы, а не новое множество записей.

Чтобы показать, как модифицировать содержимое существующей базы данных с помощью только запроса ExecuteNonQuery(), следующим шагом будет создание собственной библиотеки доступа к данным, в которой инкапсулируется процесс работы с базой данных AutoLot.

В реальной производственной среде ваша логика ADO.NET почти наверняка будет изолирована в .dll-сборке .NET по одной простой причине — повторное использование кода! В предыдущих статьях это не было сделано, чтобы не отвлекать вас от решаемых задач. Но было бы лишними затратами времени разрабатывать ту же самую логику подключения, ту же самую логику чтения данных и ту же самую логику выполнения команд для каждого приложения, которому понадобится работать с базой данных AutoLot.

В результате изоляции логики доступа к данным в кодовой библиотеке .NET различные приложения с любыми пользовательскими интерфейсами (консольный, в стиле рабочего стола, в веб-стиле и т.д.) могут обращаться к существующей библиотеке даже независимо от языка. И если разработать библиотеку доступа к данным на C#, то другие программисты в .NET смогут создавать свои пользовательские интерфейсы на любом языке (например, VB или C++/CLI).

Наша библиотека доступа к данным (AutoLotDAL.dll) будет содержать единое пространство имен (AutoLotConnectedLayer), которое будет взаимодействовать с базой AutoLot с помощью подключенных типов ADO.NET.

Начните с создания нового проекта библиотеки классов (C# Class Library) по имени AutoLotDAL (сокращенно от 'AutoLot Data Access Layer" — "Уровень доступа к данным AutoLot"), а затем смените первоначальное имя файла C#-кода на AutoLotConnDAL.cs.

Потом переименуйте область действия пространства имен в AutoLotConnectedLayer и измените имя первоначального класса на InventoryDAL, т.к. этот класс будет определять различные члены, предназначенные для взаимодействия с таблицей Inventory базы данных AutoLot. И, наконец, импортируйте следующие пространства имен .NET:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Data.SqlClient;

namespace AutoLotConnectedLayer
{
   public class InventoryDAL
   {
   }
}
```

3.1 Добавление логики подключения

Первая наша задача — определить методы, позволяющие вызывающему процессу подключаться к источнику данных с помощью допустимой строки подключения и отключаться от него. Поскольку в нашей сборке AutoLotDAL.dll будет жестко закодировано использование типов класса System.Data.SqlClient, определите приватную переменную SqlConnection, которая будет выделяться при создании объекта InventoryDAL.

Кроме того, определите метод OpenConnection(), а затем еще CloseConnection(), которые будут взаимодействовать с этой переменной:

```
public class InventoryDAL
   {
      private SqlConnection connect = null;
```

```
        public void OpenConnection(string connectionString)
        {
           connect = new SqlConnection(connectionString);
           connect.Open();
        }

        public void CloseConnection()
        {
           connect.Close();
        }
    }
```

Для краткости тип InventoryDAL не будет проверять все возможные исключения, и не будет генерировать пользовательские исключения при возникновении различных ситуаций (например, когда строка подключения неверно сформирована). Однако при создании производственной библиотеки доступа к данным вам наверняка пришлось бы задействовать технику структурированной обработки исключений, чтобы учитывать все аномалии, которые могут возникнуть во время выполнения.

3.2 Добавление логики вставки

Вставка новой записи в таблицу Inventory сводится к форматированию SQL-оператора Insert (в зависимости от введенных пользователем данных) и вызову метода ExecuteNonQuery() с помощью объекта команды. Для этого добавьте в класс InventoryDAL общедоступный метод InsertAuto(), принимающий четыре параметра, которые соответствуют четырем столбцам таблицы Inventory (CarID, Color, Make и PetName). На основании этих аргументов сформируйте строку для добавления новой записи. И, наконец, выполните SQL-оператор с помощью объекта SqlConnection:

```
public void InsertAuto(int id, string color, string make, string petName)
{
        // Оператор SQL
        string sql = string.Format("Insert Into Inventory" +
             "(CarID, Make, Color, PetName) Values(@CarId, @Make, @Color, @PetName)");

        using (SqlCommand cmd = new SqlCommand(sql, this.connect))
        {
           // Добавить параметры
           cmd.Parameters.AddWithValue("@CustID", id);
           cmd.Parameters.AddWithValue("@Make", make);
           cmd.Parameters.AddWithValue("@Color", color);
           cmd.Parameters.AddWithValue("@PetName", petName);

           cmd.ExecuteNonQuery();
        }
}
```

Определение классов, представляющих записи в реляционной базе данных — распространенный способ создания библиотеки доступа к данным. Вообще-то, ADO.NET Entity Framework автоматически генерирует строго типизированные классы, которые позволяют взаимодействовать с данными базы. Кстати, автономный уровень ADO.NET генерирует строго типизированные объекты DataSet для представления данных из заданной таблицы в реляционной базе данных.

Создание оператора SQL с помощью конкатенации строк может оказаться опасным с точки зрения безопасности (вспомните атаки вставкой в SQL). Текст команды лучше создавать с помощью параметризованного запроса, который будет описан чуть позже.

Добавление логики удаления

Удаление существующей записи не сложнее вставки новой записи. В отличие от кода InsertAuto(), будет показана одна важная область try/catch, которая обрабатывает возможную ситуацию, когда выполняется попытка удаления автомобиля, уже заказанного кем-то из таблицы Customers. Добавьте в класс InventoryDAL следующий метод:

```
public void DeleteCar(int id)
    {
      string sql = string.Format("Delete from Inventory where CarID = '{0}'", id);
      using (SqlCommand cmd = new SqlCommand(sql, this.connect))
      {
        try
        {
          cmd.ExecuteNonQuery();
        }
        catch (SqlException ex)
        {
          Exception error = new Exception("К сожалению, эта машина заказана!", ex);
          throw error;
        }
      }
    }
```

3.3 Добавление логики изменения

Когда дело доходит до обновления существующей записи в таблице Inventory, то сразу же возникает очевидный вопрос: что именно можно позволить изменять вызывающему процессу: цвет автомобиля, дружественное имя, модель или все сразу? Один из способов максимального повышения гибкости — определение метода, принимающего параметр типа string, который может содержать любой оператор SQL, но это, по меньшей мере, рискованно.

В идеале лучше иметь набор методов, которые позволяют вызывающему процессу изменять записи различными способами. Однако для нашей простой библиотеки доступа к данным мы определим единый метод, который позволяет вызывающему процессу изменить дружественное имя указанного автомобиля:

```
public void UpdateCarPetName(int id, string newpetName)
{
  string sql = string.Format("Update Inventory Set PetName = '{0}' Where CarID = '{1}'",
      newpetName, id);
  using (SqlCommand cmd = new SqlCommand(sql, this.connect))
  {
    cmd.ExecuteNonQuery();
  }
}
```

3.4 Добавление логики выборки

Теперь необходимо добавить метод для выборки записей. Как было показано ранее, объект чтения данных конкретного поставщика данных позволяет выбирать записи с помощью курсора, допускающего только чтение в прямом направлении. Посредством вызова метода Read() можно

обработать каждую запись поочередно. Все это замечательно, но теперь необходимо разобраться, как возвратить эти записи вызывающему уровню приложения.

Одним из подходов может быть получение данных с помощью метода Read() с последующим заполнением и возвратом многомерного массива (или другого объекта вроде обобщенного List<T>).

Еще один способ — возврат объекта System.Data.DataTable, который вообще-то принадлежит автономному уровню ADO.NET. DataTable — это класс, представляющий табличный блок данных (наподобие бумажной или электронной таблицы).

Класс DataTable содержит данные в виде коллекции строк и столбцов. Эти коллекции можно заполнять программным образом, но в типе DataTable имеется метод Load(), который может автоматически заполнять их с помощью объекта чтения данных! Вот пример, где данные из таблицы Inventory возвращаются в виде DataTable:

```
public DataTable GetAllInventoryAsDataTable()
{
    DataTable inv = new DataTable();
    string sql = "Select * From Inventory";
    using (SqlCommand cmd = new SqlCommand(sql, this.connect))
    {
        SqlDataReader dr = cmd.ExecuteReader();
        inv.Load(dr);
        dr.Close();
     }
  return inv;
}
```

Работа с параметризованными объектами команд

Пока в логике вставки, изменения и удаления для типа InventoryDAL мы использовали жестко закодированные строковые литералы для каждого SQL-запроса. Вы, видимо, знаете о существовании параметризованных запросов, которые позволяют рассматривать параметры SQL как объекты, а не просто кусок текста.

Работа с SQL-запросами в более объектно-ориентированной манере не только помогает сократить количество опечаток (при наличии строго типизированных свойств), ведь параметризованные запросы обычно выполняются значительно быстрее запросов в виде строковых литералов, поскольку они анализируются только один раз (а не каждый раз, как это происходит, если свойству CommandText присваивается SQL-строка). Кроме того, параметризованные запросы защищают от атак внедрением в SQL (широко известная проблема безопасности доступа к данным).

Для поддержки параметризованных запросов объекты команд ADO.NET поддерживают коллекцию отдельных объектов параметров. По умолчанию эта коллекция пуста, но в нее можно занести любое количество объектов параметров, которые соответствуют *параметрам-заполнителям (placeholder parameter)* в SQL-запросе. Если нужно связать параметр SQL-запроса с членом коллекции параметров некоторого объекта команды, поставьте перед параметром SQL символ @ (по крайней мере, при работе с Microsoft SQL Server, хотя не все СУБД поддерживают это обозначение).

Задание параметров с помощью типа DbParameter

Прежде чем приступить к созданию параметризованных запросов, ознакомимся с типом DbParameter (базовый класс для объектов параметров поставщиков). У этого класса есть ряд свойств, которые позволяют задать имя, размер и тип параметра, а также другие характеристики, например, направление просмотра параметра. Некоторые важные свойства типа DbParameter приведены ниже:

DbType

Выдает или устанавливает тип данных из параметра, представляемый в виде типа CLR

Direction

Выдает или устанавливает вид параметра: только для ввода, только для вывода, для ввода и для вывода или параметр для возврата значения

IsNullable

Выдает или устанавливает, может ли параметр принимать пустые значения

ParameterName

Выдает или устанавливает имя DbParameter

Size

Выдает или устанавливает максимальный размер данных для параметра (полезно только для текстовых данных)

Value

Выдает или устанавливает значение параметра

Для демонстрации заполнения коллекции объектов команд совместимыми с DBParameter объектами переделаем метод InsertAuto() так, что он будет использовать объекты параметров (аналогично можно переделать и все остальные методы, но нам будет достаточно и настоящего примера):

```
public void InsertAuto(int id, string color, string make, string petName)
      {
        // Оператор SQL
        string sql = string.Format("Insert Into Inventory" +
          "(CarID, Make, Color, PetName) Values('{0}','{1}','{2}','{3}')", id, make, color,
petName);

        // Параметризованная команда
        using (SqlCommand cmd = new SqlCommand(sql, this.connect))
        {
          SqlParameter param = new SqlParameter();
          param.ParameterName = "@CarID";
          param.Value = id;
          param.SqlDbType = SqlDbType.Int;
          cmd.Parameters.Add(param);

          param = new SqlParameter();
          param.ParameterName = "@Make";
          param.Value = make;
          param.SqlDbType = SqlDbType.Char;
          param.Size = 10;
          cmd.Parameters.Add(param);

          param = new SqlParameter();
          param.ParameterName = "@Color";
          param.Value = color;
          param.SqlDbType = SqlDbType.Char;
          param.Size = 10;
          cmd.Parameters.Add(param);

          param = new SqlParameter();
          param.ParameterName = "@PetName";
          param.Value = petName;
          param.SqlDbType = SqlDbType.Char;
          param.Size = 10;
          cmd.Parameters.Add(param);
```

```
            cmd.ExecuteNonQuery();
        }
    }
```

Обратите внимание, что здесь SQL-запрос также содержит четыре символа-заполнителя, перед каждым из которых находится символ @. С помощью свойства ParameterName в типе SqlParameter можно описать каждый из этих заполнителей и задать различную информацию (значение, тип данных, размер и т.д.), причем строго типизированным образом. После подготовки всех объектов параметров они добавляются в коллекцию объекта команды с помощью вызова Add().

Для оформления объектов параметров здесь используются различные свойства. Однако учтите, что объекты параметров поддерживают ряд перегруженных конструкторов, которые позволяют задавать значения различных свойств (что дает более компактную кодовую базу). Учтите также, что в Visual Studio 2010 имеются различные графические конструкторы, которые автоматически создадут за вас большой объем этого утомительного кода работы с параметрами.

Создание параметризованного запроса часто приводит к большему объему кода, но в результате получается более удобный способ для программной настройки SQL-операторов, а также более высокая производительность. Эту технику можно применять для любых SQL-запросов, хотя параметризованные запросы наиболее удобны, если нужно запускать хранимые процедуры.

Практическое занятие № 26-27 Разработка индивидуальной информационной системы

Этапы выполнения работы:

1. Разработать интерфейс по теме курсового проекта.
2. Добавить кнопки, позволяющие вносить данные, сохранять их, редактировать, производить выборку, удалять, осуществлять поиск.

Список использованных источников

1 Вставка, удаление, обновление записей в базе данных. http://professorweb.ru/my/ADO_NET/base/level1/1_7.php

2 Справочное руководство по MySQL. http://www.mysql.ru/docs/man/INSERT.html