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
МДК.05.02 Разработка кода информационных систем

5 семестр

Специальность

09.02.07 Информационные системы и программирование

Ростов-на-Дону

2023

Методические рекомендации по МДК.05.02 Разработка кода информационных систем разработаны с учетом ФГОС среднего профессионального образования специальности 09.02.07 Информационные системы и программирование, предназначены для студентов и преподавателей колледжа.

Методические рекомендации содержат цели изучения курса, планируемые результаты, необходимый теоретический материал, примеры выполнения заданий, а также список литературы.

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

Рассмотрены на заседании цикловой комиссии специальностей 09.02.04 Информационные системы (по отраслям), 09.02.05 Прикладная информатика (по отраслям) и 09.02.07 Информационные системы и программирование

Протокол № 9 от «30» июня 2023 г

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

и одобрены решением учебно-методического совета колледжа.

Протокол № _7 от «30» июня 2023 г

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

Рекомендованы к практическому применению в образовательном процессе.

Содержание

Проектная документация на разработку ИС 4
Практическое занятие № 1-2 Сбор сведений для разработки ИС. Изучение ГОСТа ТЗ на разработку ИС 4
Интерфейс пользователя 4
Практическое занятие № 3-4 Проектирование интерфейса пользователя 4
Разработка однопользовательской АИС средствами Visual Studio и СУБД Access 6
Практическое занятие № 5-6 Проектирование и разработка АИС 6
Проектирование и разработка АИС 10
Практическое занятие № 7 Создание базы данных, основы работы с таблицами 10
Практическое занятие № 8 Работа с таблицами 15
Практическое занятие № 9 Логические операторы. Команды обработки данных 17
Часть 1 Логические операторы 17
Часть 2 Команды обработки данных 21
Практическое занятие № 10 Математические функции. Работа с датой и временем. 27
Часть 1 Математические функции 27
Часть 2 Работа с датой и временем 46
Проектирование серверной части многопользовательских информационных систем ... 50
Практическое занятие № 11 Концептуальное, логическое, физическое проектирование 50
Практическое занятие № 12 Создание серверного приложения преобразованием проекта базы данных формата MS Access в формат Sql-Server 54
Практическое занятие № 13 Хранимые процедуры 55
Практическое занятие № 13 Хранимые процедуры 64
Практическое занятие № 14 Создание триггеров 74

Проектная документация на разработку ИС

Практическое занятие № 1-2 Сбор сведений для разработки ИС. Изучение ГОСТа ТЗ на разработку ИС

Цель: формирование знание ГОСТа ТЗ на разработку ИС, умений - осуществлять постановку задач по обработке информации; проводить анализ предметной области; формирование компетенции по сбору исходных данных для разработки проектной документации на информационную систему.

Задание

1 Сделать анализ предметной области по индивидуальной теме (Приложение А).
2 Изучить ГОСТ 34.602-2020, создать ТЗ.
3 Сдать преподавателю итог работы.

Интерфейс пользователя

Практическое занятие № 3-4 Проектирование интерфейса пользователя

Цель: формировать компетенции по разработке проектной документации на разработку информационной системы в соответствии с требованиями заказчика;

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Интерфейс — это вся видимая пользователю часть сервиса, с которой он взаимодействует, решая свои задачи.

Проектирование интерфейсов — это всегда поиск наиболее эффективного решения, которое основано на понимании задач, мотиваций и обстоятельств пользователей и в то же время учитывает цели, возможности и ограничения со стороны бизнеса и технологий.

Результатом проектирования являются макеты и сопроводительные материалы, которые можно передать команде разработки.

Алгоритм проектирования интерфейса пользователя

Этап	Содержание этапа
1. Исследовать	Детально изучить сам сервис и все, что его окружает: поведение пользователей и сценарии их взаимодействия с сервисом, прямых и косвенных конкурентов, возможные риски и ограничения, релевантные технологии и тренды индустрии. **Источники информации:** бриф заказчика, конкурентный анализ, интервью с экспертами в предметной области, кабинетное и явное обследование, юзабилити-тесты и эксперименты.
2. Моделировать	Разделить модели пользователей, описать их различия в целях, ожиданиях и контекстах использования сервиса.

	Сформировать требования к каждому разделу сервиса: на какие вопросы пользователи должны получить ответы, необходимые функции, технологические и бизнес-требования.
3. Синтезировать	Построить Customer Journey Map: схематично визуализируем логику взаимодействия пользователей с сервисом при решении разных задач. Описать User Stories: детализировать в текстовом формате, как пользователь закрывает свои потребности через интерфейс сервиса, и как интерфейс реагирует на действия пользователя.
4. Детализировать	На основе зафиксированных ожиданий и адаптированных визуальных паттернов из брендбука компании разработать визуальную концепцию сервиса — несколько ключевых экранов, иллюстрирующих, как будет выглядеть интерфейс продукта, и объяснить заложенные в концепцию принципы. Итеративно детализировать пользовательские сценарии и презентовать заказчику. Собрать обратную связь и внести корректировки. Систематизировать дизайн-макеты, используя библиотеки стилей и компонентов — это помогает ускорить дизайн-процесс и упростить разработку продукта.
5. Тестировать	Собрать макеты в интерактивный прототип, который моделирует реакции интерфейса на поведение пользователя. Провести usability-тестирование прототипа на респондентах и подтвердить или опровергнуть гипотезы, которые были заложены в дизайн-решениях. Доработать макеты по результатам тестирования.
6. Внедрить	Консультация команды разработки на протяжении всего процесса проектирования — объяснение сложных механик, взаимосвязи и паттернов поведения элементов. При необходимости корректировать решения и обновить документацию, чтобы обеспечить концептуальную целостность проектного решения в условиях меняющихся технологических ограничений.

Результат

Макеты	Файл в Figma, содержащий детализированные макеты по всем ключевым сценариям, и UI-kit.
Отчет о usability-тестировании	Файл в PDF, содержащий ключевые результаты тестирования, описание пользователей и выявленные проблемы.
Промежуточные артефакты	Документы, на основе которых были созданы макеты: CJM, User Stories и другие материалы.

Задание

1 Спроектировать интерфейс пользователя для индивидуальной ИС.

2 Сдать преподавателю результат работы.

Разработка однопользовательской АИС средствами Visual Studio и СУБД Access

Практическое занятие № 5-6 Проектирование и разработка АИС

Цель: формирование умения использовать алгоритмы обработки информации для различных приложений; изучение механизма связывания источника данных с элементом управления dataGridView; формирование компетенций при построении приложений по обработке информационных массивов с применением элементов управления: DataSet, DataGridView, BindingSourse, BindingNavigator.

Задание

1 По разработанному ТЗ создать ИС в среде VisualStudio.

2 Сдать преподавателю итог работы.

Содержание занятия:

1.Создание пользовательского интерфейса.

2. Создание модулей для обработки кнопок ввода, редактирования и поиска данных.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

DataAdapter является средством связи между Dataset и Connection. С Dataset он связан посредством команд:

Fill - загрузить данные,

Update - записать изменения.

В момент записи создаётся список ошибок (Error Collection). DataAdapter работает с Connection с помощью реляционного метода. Команды взаимодействия с БД (Command) можно создавать как вручную, так и автоматически.

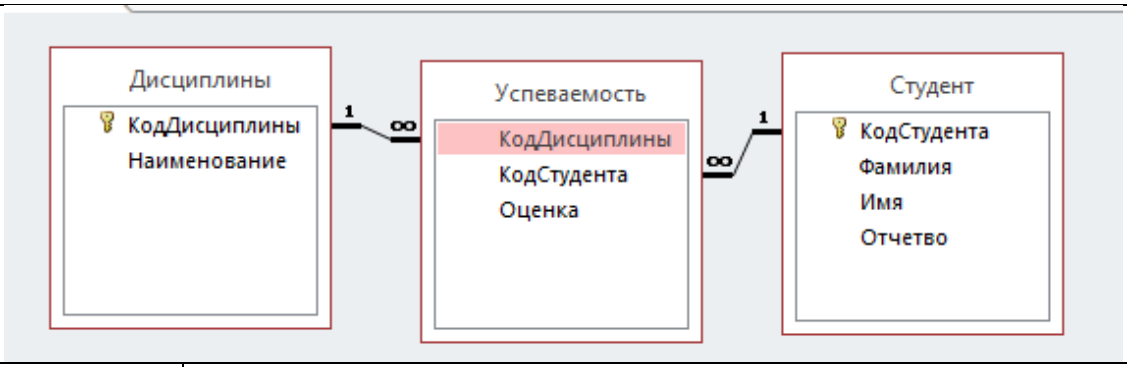
ХОД РАБОТЫ

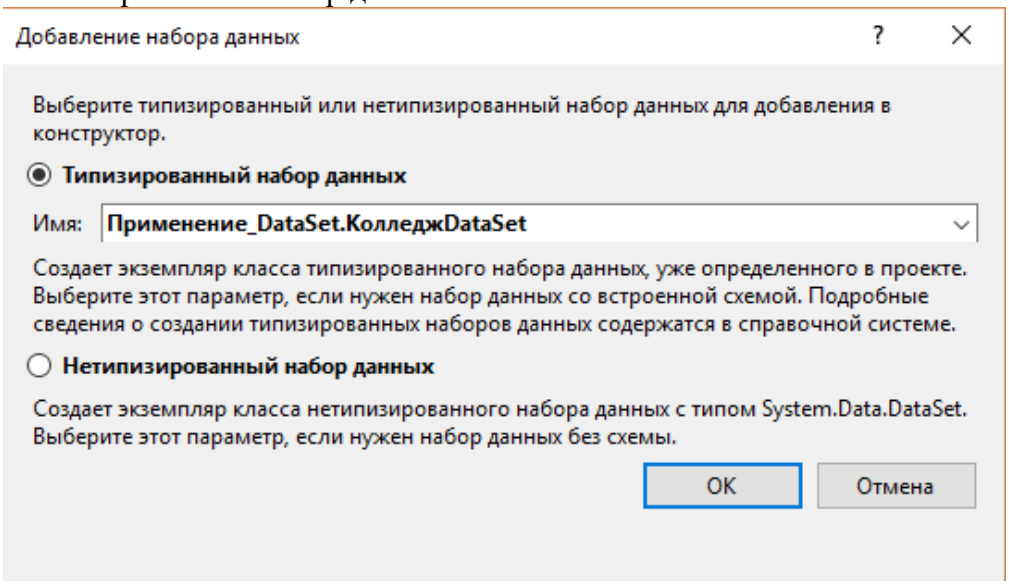
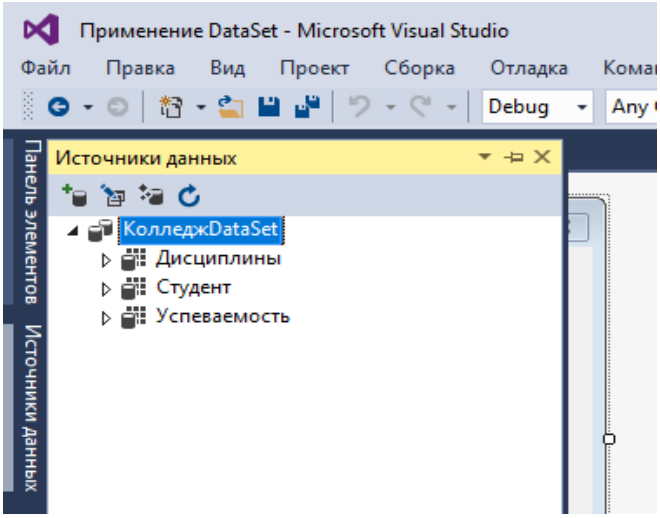
Задание 1. Создать приложение, позволяющее извлекать данные с помощью объекта DataSet.

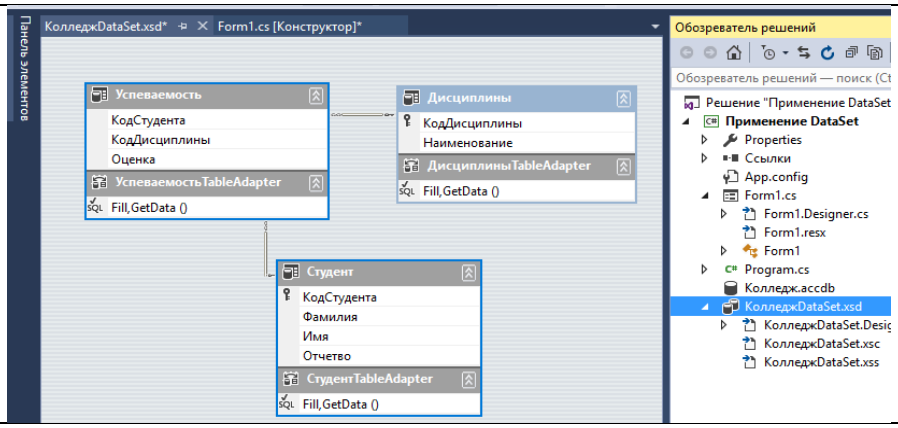
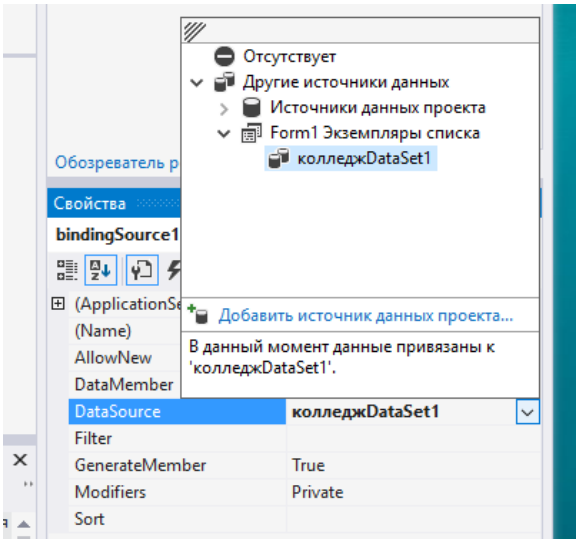
Примечание: в качестве альтернативы для заполнения элемента управления вручную, можно задать свойства для привязки DataGridView в данных источника и автоматически заполняет ее данными.

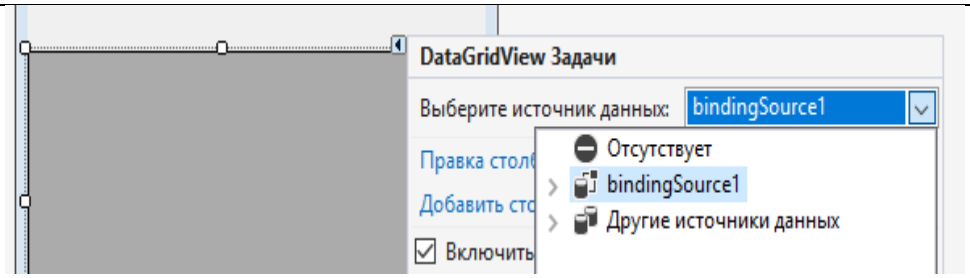
Алгоритм создания приложения:

В качестве источника данных выбрать базу данных Колледж.accdb, созданную в MS Access. Для обработки данных в приложении использовать компоненты DataGridView, BindingSourse, BindingNavigator, DataSet.
Таблицы базы данных



1. Выбрать источник данных	**Проект / Добавить новый источник данных /** указать путь к базе данных, выбрать таблицы и требуемые элементы таблицы: в окне Мастер настройки источника данных выбрать База данных / Набор данных / Создать подключение / Файл базы данных Microsoft Access / Обзор / Выбрать базу / Проверить подключение / Ok / Далее/ Да/ Далее/ Поставить галочку Таблицы / Готово
2. Добавить к своему проекту компонент DataSet	При добавлении Dataset, в мастере необходимо выбрать пункт «Типизированный набор данных».  Если все шаги были проделаны верно, то появится КолледжDataSet в источниках данных  Дважды щелкните на файле КолледжDataSet.xsd, появится схема данных.

	
3. На форму из вкладки «Данные» панели элементов добавить компоненты	Добавьте: BindingSource, DataGridView, BindingNavigator (два последних являются визуальными компонентами).
4. Настроить свойства компонента bindingSource1	1) в элементе bindingSource1 выбрать в свойстве DataSource именно тот Dataset, который расположен на одной форме с этим компонентом  2) в поле DataMember нужно выбрать имя таблицы, связанной с этим компонентом - Студент. Если всё сделано, верно, то в списке невидимых компонентов должен появиться TableAdapter - студентTableAdapter. 3) в редакторе кода найдите строку, которая загружает данные в адаптер таблиц. Этот код был сгенерирован при установке привязки данных. Код должен иметь следующий вид: `this.студентTableAdapter.Fill(this.колледжDataSet1.Студент);` Он находится в событии формы `Form1_Load`.
5. Настроить свойства компонента dataGridView1	В dataGridView1 нужно свойству DataSource установить значение bindingSource1.

	 Появятся заголовки столбцов таблицы.
6. Настроить свойства компонента bindingNavigator1	Установить свойству BindingSource соответствующий компонент формы - bindingSourse1

Задание 2. Запрограммировать кнопки навигационной панели.
Примечание: все операции над записями в программе происходят через BindingSource.

Алгоритм создания приложения:

1. Создайте обработчик события для кнопки ToolStripButton «**Загрузить**»	Синтаксис команды: TableAdapterName.Fill (DataSetName.TableName); Применительно к нашему проекту команда имеет следующий вид: `this.студентTableAdapter.Fill(this.колледжDataSet1.Студент);`
2. Создайте обработчик события для кнопки «**Сохранить**»	В некоторых случаях компонент BindingNavigator уже содержит кнопку **Сохранить**, но код, сгенерированный конструктором Windows Forms, при этом отсутствует. В этом случае приведенный код можно поместить в Click обработчик событий кнопки, вместо создания полностью новой кнопки ToolStrip. Кнопка по умолчанию отключена, свойству Enabled кнопки необходимо присвоить значение **true**, чтобы кнопка работала правильно. Синтаксис команды: TableAdapterName.Update(DataSetName.TableName); Добавим код: `this.студентTableAdapter.Update(this.колледжDataSet1);`
3. Создайте обработчик события для кнопки ToolStripButton «**Отмена**»	BindingSourceName.CancelEdit(); Метод CancelEdit распространяется на строку данных. Сохраните все изменения, сделанные при просмотре этой отдельной записи, до перехода к следующей записи.
4. Создайте обработчик события для кнопки ToolStripButton «**Найти**»	Фильтрация задается по следующему шаблону: bindingSource1.Filter = "Условие фильтра". Добавим код: `bindingSource1.Filter = "Фамилия like '" + textBox1.Text + "%'";`
Элемент TableAdapter предназначен для заполнения dataSet формы с помощью команды Fill, а также для сохранения этого набора данных в БД. Кроме того, с помощью этого	

компонента можно создавать команды для добавления записей - `TablAdapter.Insert()` - и их удаления – `TablAdapter.Delete()`.

1. **Сортировка:**

   bindingSource1.Sort ="Поле таблицы" asc/desc;

   asc/desc - по возрастанию или убыванию
2. **Перемещение по записям:**

   bindingSource1.MoveNext(); - следующий элемент

   bindingSource1.MovePrevious(); - предыдущий элемент
3. **Удалить запись в БД:**

   tableAdapter.Update(this.DataSet);

```
студентTableAdapter.Delete(1,"Хромов","","");
```

Перед изменением данных необходимо выполнить команду:
bindingSource1.EndEdit(); - применить незавершенные изменения к базовому источнику данных.

Проектирование и разработка АИС

Практическое занятие № 7 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.

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. `Query OK, 1 row affected (0.00 sec)` 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.
	`+------------+`
	`| Data base |`
	`+------------+`
	`| employees |`
	`| mysql |`
	`| test |`
	`+------------+`
	`2 rows in set (0.00 sec)`
	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),`

<table>
<tr><td></td><td></td><td>

```
age int,
yos int,
salary int,
perks int,
email varchar(60)
);
```

</td></tr>
<tr><td>9.</td><td colspan="2">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.</td></tr>
<tr><td>10.</td><td>Для работы с БД, необходимо её "активировать" или "выбрать". В приглашении mysql выполните команду:</td><td>

```
SELECT DATABASE();
```

</td></tr>
<tr><td>11.</td><td>Просмотр таблиц в базе</td><td>

```
mysql> SHOW TABLES;
+----------------------+
| Tables in employees |
+----------------------+
|   employee data      |
+----------------------+
1 rows in set (0.00 sec)
```

</td></tr>
<tr><td>12.</td><td>Заполнить таблицу данными</td><td>

```
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.</td></tr>
<tr><td>13.</td><td>Измените тип поля title на Next</td><td>Alter Table имя_таблицы Modify имя-поля новый_тип</td></tr>
</table>

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 № 8 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.

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.

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. Что делает следующий оператор?

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

4. Выведите данные о сотрудниках, отсортировав их по возрасту (по убыванию).
5. Выведите данные о сотрудниках, отсортировав их по зарплате (по возрастанию).
6. Создайте таблицу `users (логин, пароль).` Пароль зашифровать.

Практическое занятие № 9 Логические операторы. Команды обработки данных

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.

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:

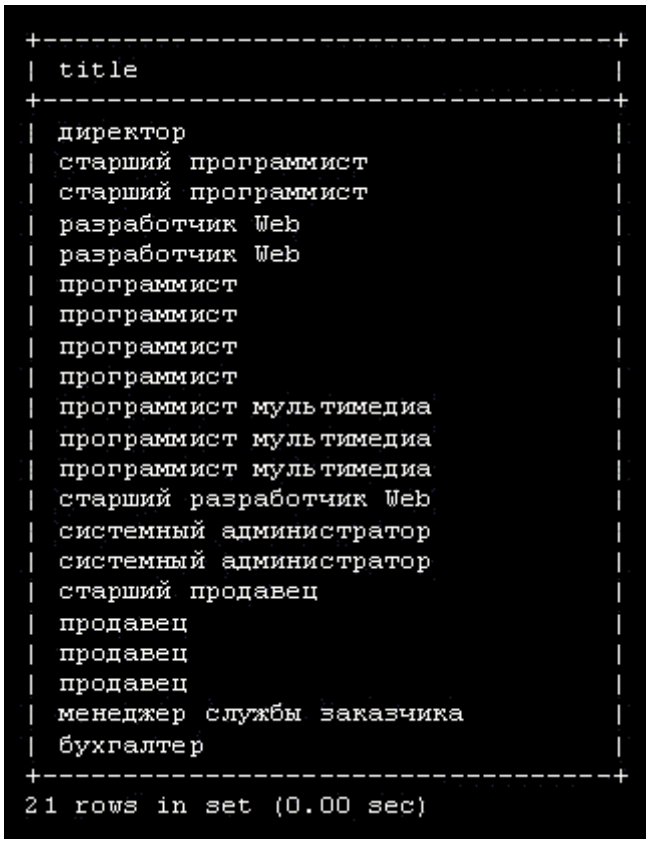
```
SELECT (что-нибудь) from таблица LIMIT начальная строка,
извлекаемое число записей;

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.

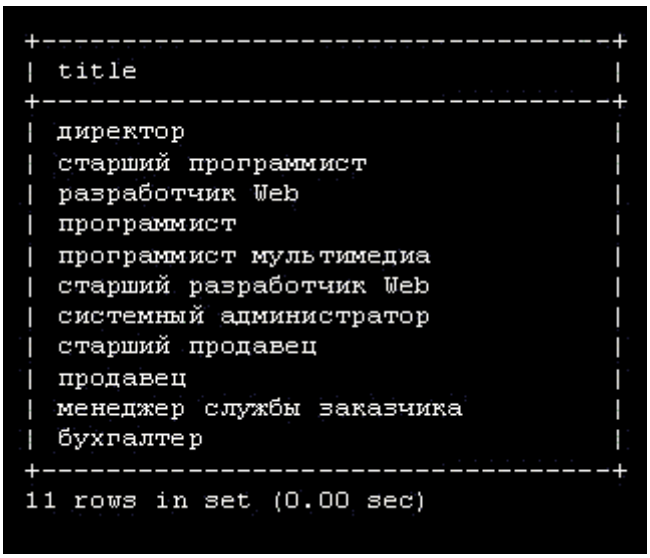


```
+-----------------------------------+
| title                             |
+-----------------------------------+
| директор                          |
| старший программист               |
| старший программист               |
| разработчик Web                   |
| разработчик Web                   |
| 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');
```

1. Вот более сложный оператор, который объединяет `BETWEEN` и `IN`. Что он делает?
2. Вывести список сотрудников (фамилию и имя), которые занимают должность "программист" или "разработчик Web" и отсортировать их фамилии по алфавиту.
3. Найдите имена 5 самых молодых сотрудников компании.

4. Извлеките 5 записей, начиная с 10 строки.
5. Выведите имя, фамилию и зарплату сотрудника, который получает самую большую зарплату.
6. Что делает следующий оператор?

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

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

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.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

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.1 приведен результат запроса.

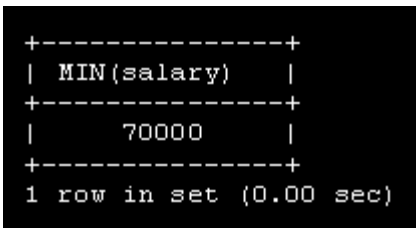


Рис. 1. Поиск минимальной зарплаты

2. Максимальное значение

select MAX(salary) from employee_data;

На рис. 2. приведен результат запроса.

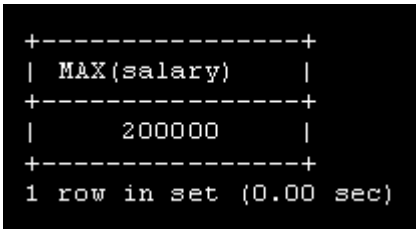


Рис. 2. Поиск максимальной зарплаты

3. 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.

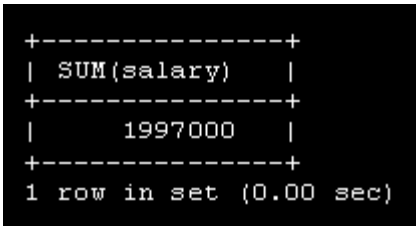


Рис. 3 Сумма всех зарплат

Аналогично можно вывести общую сумму надбавок, выдаваемых сотрудникам.

select SUM(perks) from employee_data;

На рис. 4 приведен результат запроса.

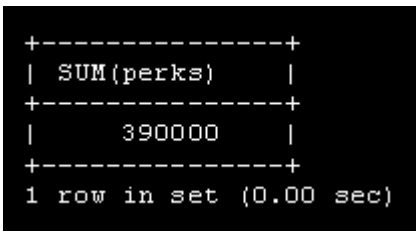


Рис. 4 Сумма всех надбавок

Можно найти также общую сумму зарплаты и надбавок.

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

На рис. 5 приведен результат запроса.

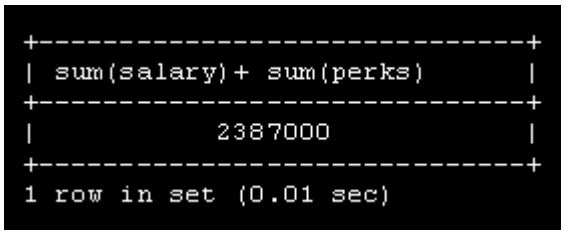


Рис. 5 Общая сумма зарплаты и надбавок

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.

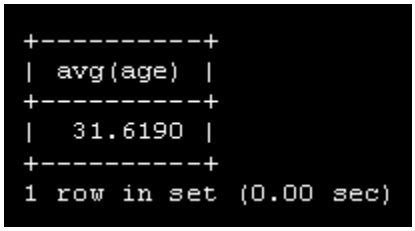


Рис. 6 Средний возраст сотрудников

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.

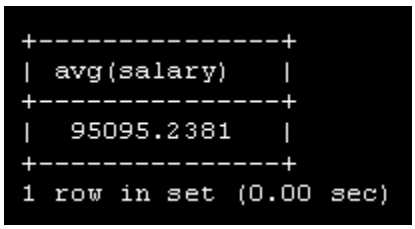


Рис. 7 Средняя зарплата сотрудников

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.

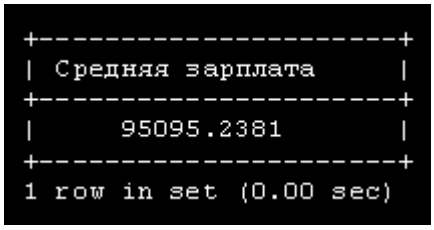


Рис. 8 Вывод средней зарплаты с использованием псевдо-имен столбцов.

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.

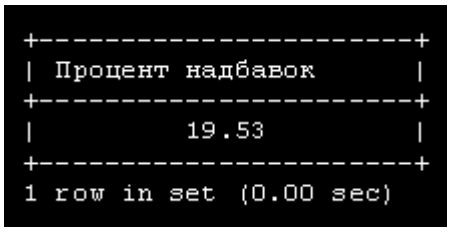


Рис. 9 Вывод процента зарплаты, которую сотрудники получают в качестве надбавок с использованием псевдо-имен

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.

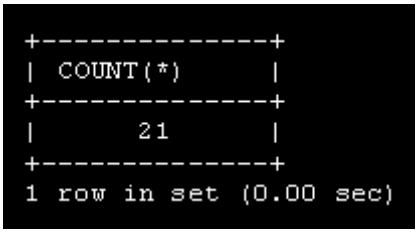


Рис. 10 Общее количество записей

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.

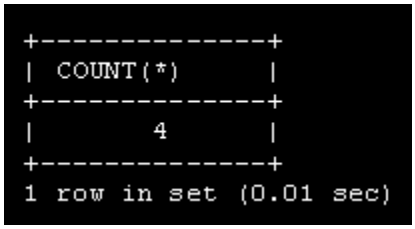


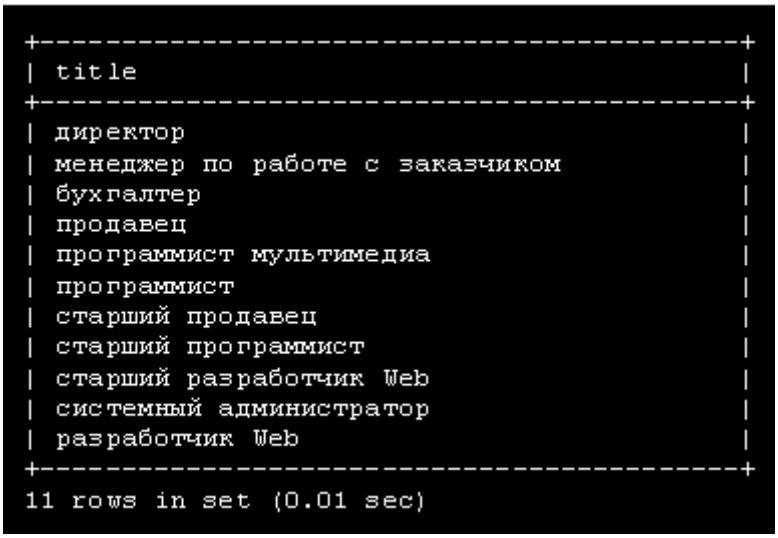
Рис 11 Общее количество сотрудников-программистов

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.



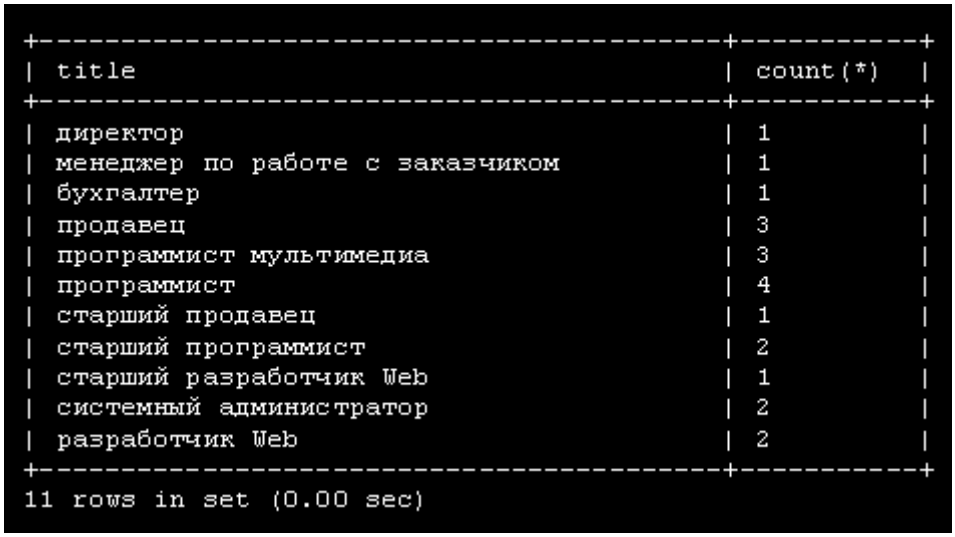
```
+-----------------------------------------+
| title                                   |
+-----------------------------------------+
| директор                                |
| менеджер по работе с заказчиком         |
| бухгалтер                               |
| продавец                                |
| 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

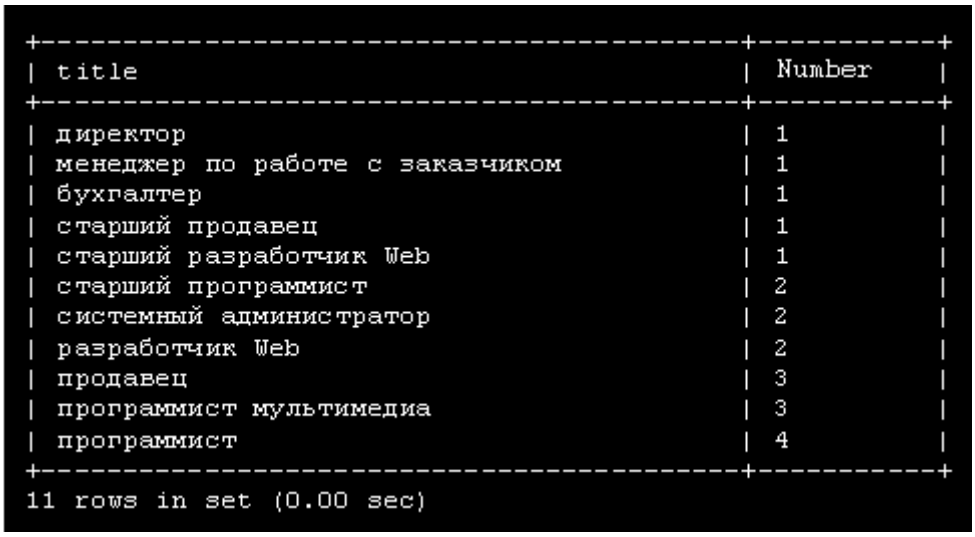
В предыдущей команде *MySQL* сначала создает группы различных должностей, а затем выполняет подсчет в каждой группе.

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 № 10 Математические функции. Работа с датой и временем.

Часть 1 Математические функции

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

Этапы выполнения работы:

1. Запустить MySql.
2. Создать запросы, указанные ниже.
3. 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. 2ф, 2б и 2в приведены примеры работы с функцией ASIN.

```
mysql> SELECT ASIN(1);

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

Рис. 2(а). Арксинус числа

```
mysql> SELECT ASIN(0.2);

+----------------------+
| ASIN(0.2)            |
+----------------------+
|       0.201358       |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 2(б). Арксинус числа

```
mysql> SELECT ASIN ('hello');

+----------------------+
| ASIN('hello')        |
+----------------------+
|       0.000000       |
+----------------------+
1 row in set (0.00 sec)
```

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.

```
mysql> SELECT ACOS(1);

+----------------------+
| ACOS(1)              |
+----------------------+
|       0.000000       |
+----------------------+
1 row in set (0.00 sec)
```

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

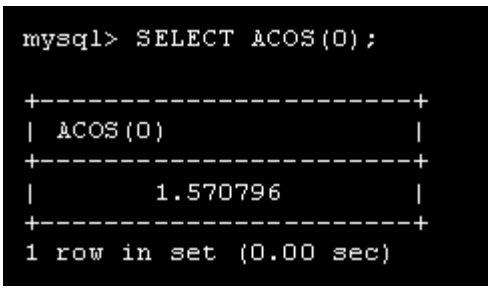


Рис. 3(в). Арккосинус числа

ATAN(число)

Эта *функция* возвращает арктангенс числа.

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

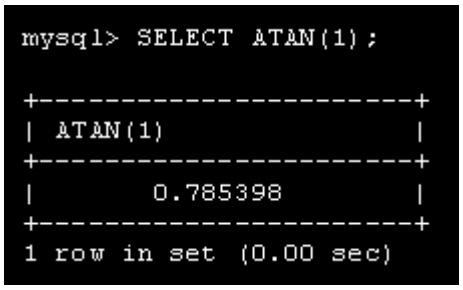


Рис. 4(а). Арктангенс числа

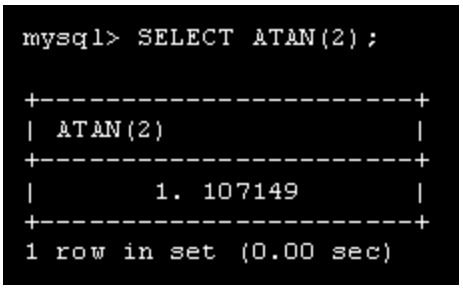


Рис. 4(б). Арктангенс числа

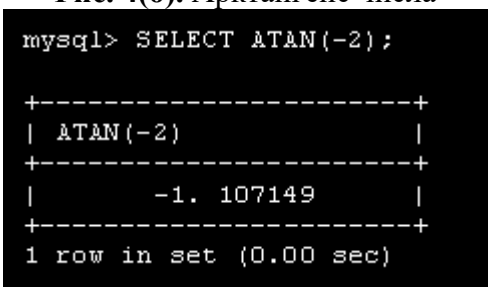


Рис. 4(в). Арктангенс числа

ATAN2(число1, число2)

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

На рис. 5а, 5б, 5в 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. 6а, 6б и 6в приведены примеры работы с функцией CEIL.

```
mysql> SELECT CEILING(1.3);

+----------------------+
| CEILING (1.3)        |
+----------------------+
|          2           |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 6(а). Функция CEIL

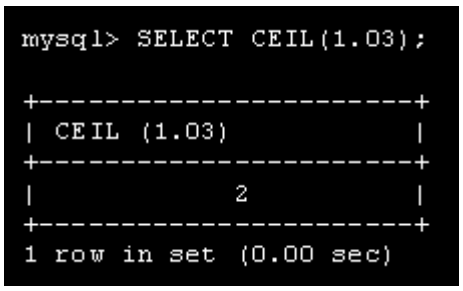


Рис. 6(б). Функция CEIL

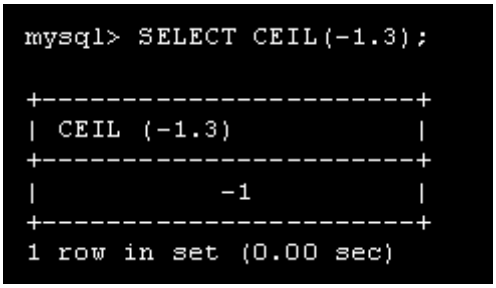


Рис. 6(в). Функция CEIL

COS(число)

Возвращает косинус числа

На рис. 7 приведен пример работы с функцией COS.

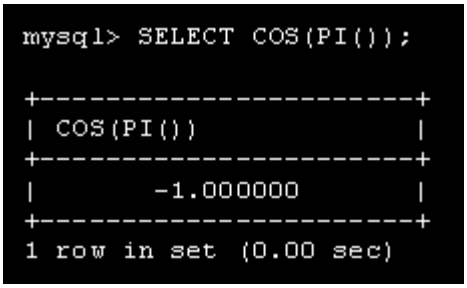


Рис. 7. Косинус числа

COT(число)

Возвращает котангенс числа.

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

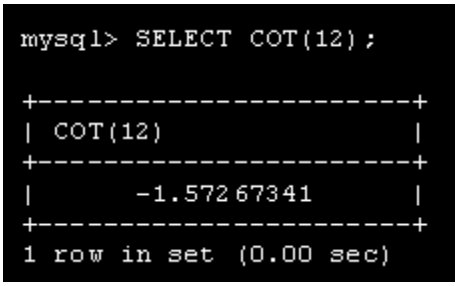


Рис. 8(а). Котангенс числа

```
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. 11а, 11б 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.12а, 12б и 12в 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. 15а, 15б приведены примеры работы с функцией LN.

```
mysql> SELECT LN(2);

+----------------------+
|         LN (2)        |
+----------------------+
|       0.693147        |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 15(а). Натуральный логарифм числа

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

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

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. 16а, 16б приведены примеры работы с функцией LOG2.

```
mysql> SELECT LOG(2, 65536);
+----------------------+
|      LOG(2, 65536)   |
+----------------------+
|       16.000000      |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 16(а). Логарифм числа по основанию

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

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. 17а и 17б 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.18а, 18б 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

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

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. 19а, 19б, 19в и 19г приведены примеры работы с функцией MOD.

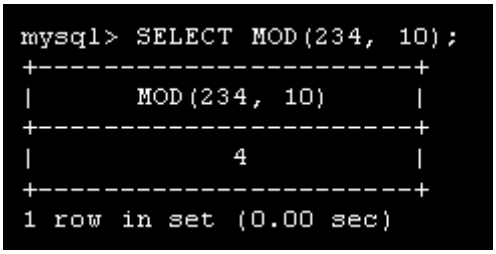


Рис. 19(а). Остаток от деления

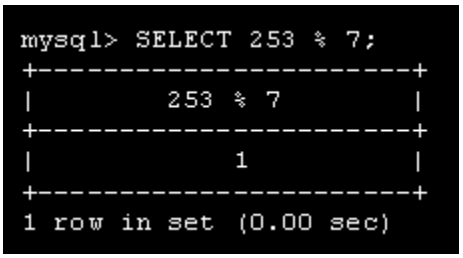


Рис. 19(б). Остаток от деления

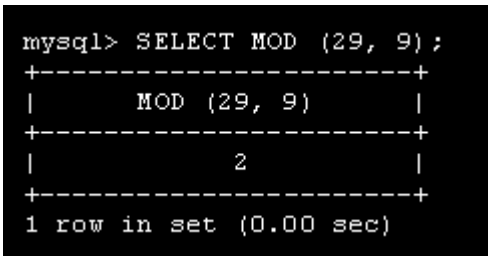


Рис. 19(в). Остаток от деления

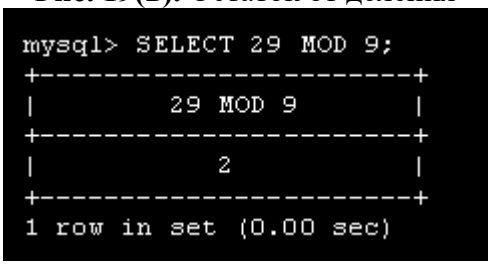


Рис. 19(г). Остаток от деления

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. 20а и 20б 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.21а, 21б и 21в 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

```
mysql> SELECT POW(2,3);
+----------------------+
|        POW(2,3)      |
+----------------------+
|        8.000000      |
+----------------------+
1 row in set (0.00 sec)
```

Рис.21(б). Возведение числа в степень

```
mysql> SELECT POWER(2,-2);
+----------------------+
|       POW(2,-2)      |
+----------------------+
|       0.250000       |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 21(в). Возведение числа в степень

RADIANS(число)

Возвращает *аргумент*, преобразованный из градусов в радианы.

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

```
mysql> SELECT RADIANS(90);
+----------------------+
|      RADIANS(90)     |
+----------------------+
|        1.570796      |
+----------------------+
1 row in set (0.00 sec)
```

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

```
mysql> SELECT RADIANS(45);
+----------------------+
|      RADIANS(45)     |
+----------------------+
|   0. 78539816339745  |
+----------------------+
1 row in set (0.00 sec)
```

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. 23а и 23б приведены примеры работы с функцией RAND.

```
mysql> SELECT RAND();
+----------------------+
|         RAND()       |
+----------------------+
|   0. 9233482386203   |
+----------------------+
1 row in set (0.00 sec)

mysql> SELECT RAND();
+----------------------+
|         RAND()       |
+----------------------+
|   0. 4738563659245   |
+----------------------+
1 row in set (0.00 sec)
```

Рис. 23(а). Создание случайных чисел

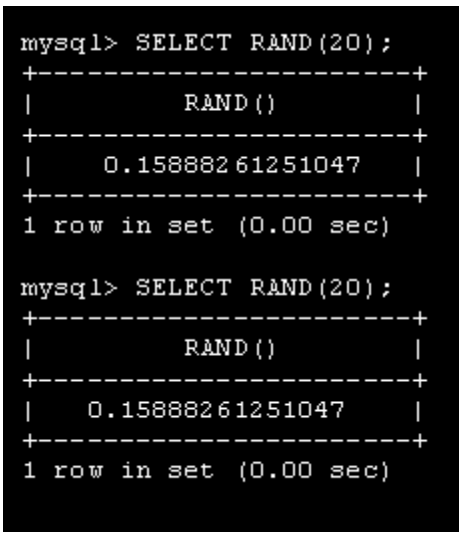


Рис. 23(б). Создание случайных чисел

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. 24а, 24б, 45в, 24г, 24д и 24е приведены примеры работы с функцией ROUND.

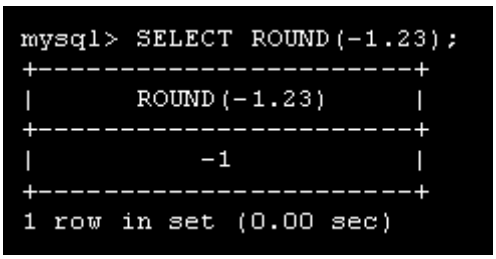


Рис. 24(а). Округление числа

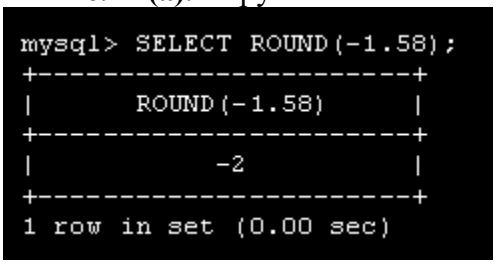


Рис. 24(б). Округление числа

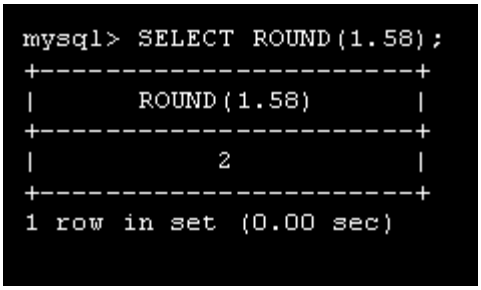


Рис. 24(в). Округление числа

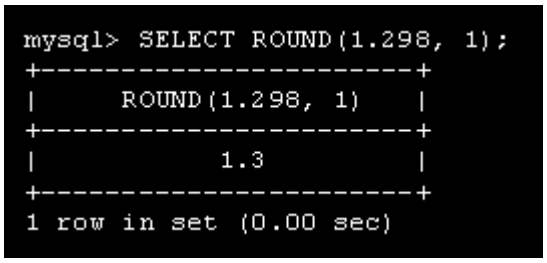


Рис. 24(г). Округление числа

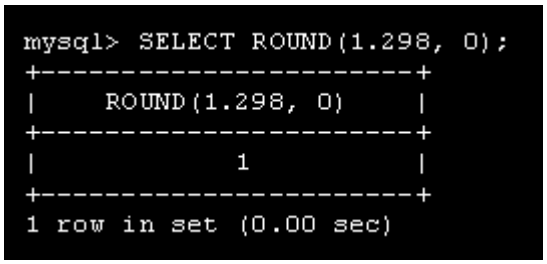


Рис. 24(д). Округление числа

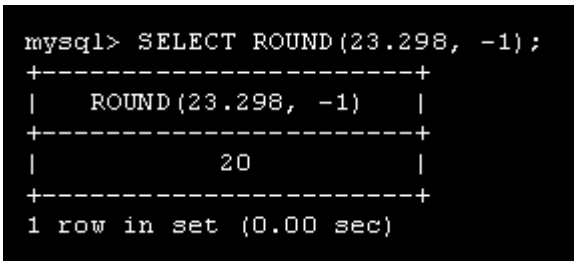


Рис. 24(е). Округление числа

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.25а, 25б и 25в 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. 26а и 26б 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. 27а, 27б и 27в приведены примеры работы с функцией SQRT.

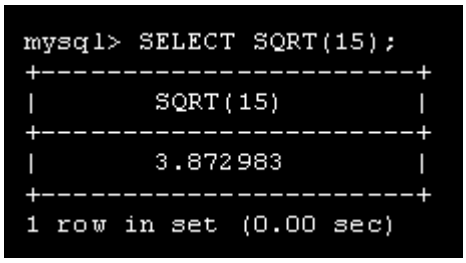


Рис. 27(а). Квадратный корень

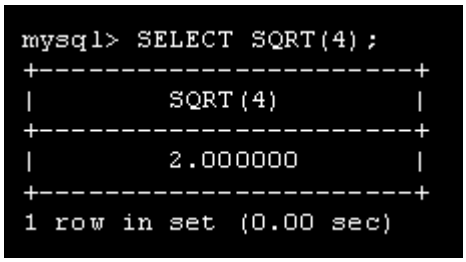


Рис. 27(б). Квадратный корень

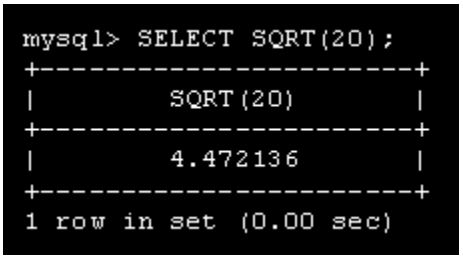


Рис. 27(в). Квадратный корень

TAN(число)

Возвращает тангенс числа.

На рис. 28 приведен пример работы с функцией TAN.

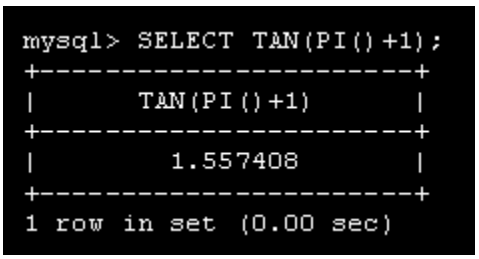


Рис. 28. Тангенс числа

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. 29а, 29б, 29в, 29г и 29д приведены примеры работы с функцией TRUNCATE.

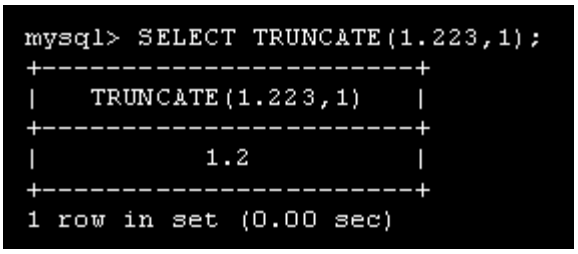


Рис. 29(а). Усечение числа

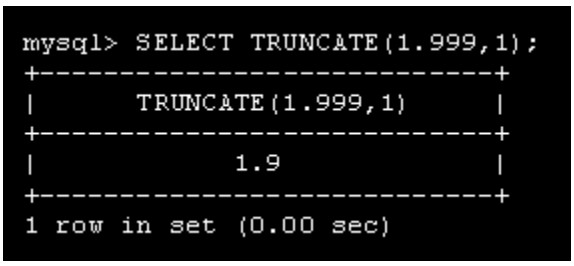


Рис. 29(б). Усечение числа

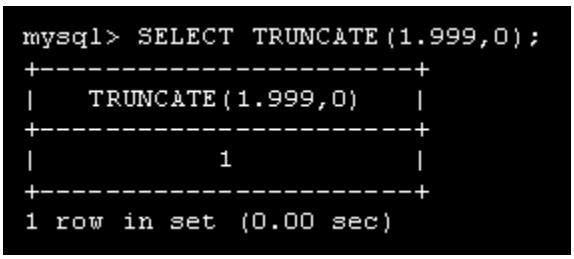


Рис. 29(в). Усечение числа

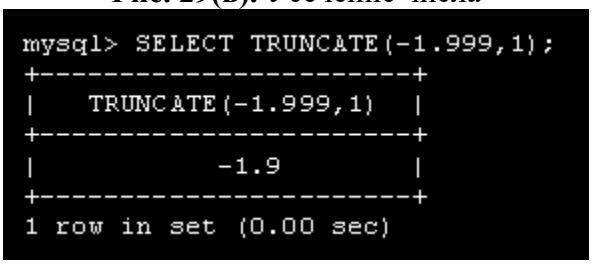


Рис. 29(г). Усечение числа

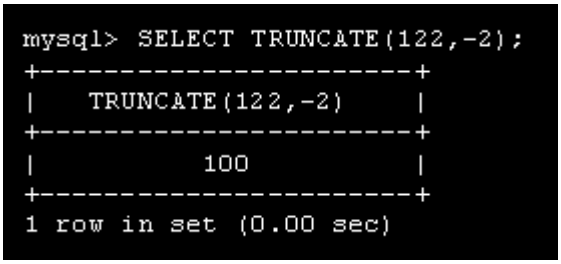


Рис. 29(д). Усечение числа

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. 29е).

```
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.

Проектирование серверной части многопользовательских информационных систем

Практическое занятие № 11 Концептуальное, логическое, физическое проектирование

Цель занятия: формирование умения – осуществлять выбор модели и средства построения информационной системы и программных средств; навыка построения схемы концептуальной, логической, физической модели.

Этапы выполнения работы:

1. Создать концептуальную модель базы данных индивидуальной ИС.
2. Создать логическую модель базы данных.
3. Создать физическую модель базы данных.
4. Разработать таблицы БД в СУБД Access.
5. Построить связи.
6. Продемонстрировать полученные модели и БД преподавателю, защитить выполненную работу.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1 Этапы проектирования БД

База данных – интеграция данных, предназначенных для решения нескольких задач разных пользователей. Должны учитываться требования к данным каждого пользователя (его представление о данных и связях между ними).

Проектирование БД – процесс создания схемы базы данных и определения необходимых ограничений целостности.

Концептуальная модель данных — схема наивысшего уровня с минимальным количеством подробностей. Достоинство этого подхода заключается в возможности отобразить общую структуру модели и всю архитектуру системы. Менее масштабные системы могут обойтись и без этой модели. В этом случае можно сразу переходить к логической модели.

Логическая модель данных содержит более подробную информацию, нежели концептуальная модель. На этом уровне определяются более подробные операционные и транзакционные сущности. Логическая модель не зависит от технологии, в которой она будет применяться.

Физическая модель данных: на основе каждой логической модели данных можно составить одну или две физических модели. В последних должно присутствовать достаточно технических подробностей для составления и внедрения самой базы данных.

2 Построение концептуальной модели данных

Концептуальная модель - это отражение предметной области, для которой разрабатывается база данных. Не вдаваясь в теорию, отметим, что это некая диаграмма с принятыми обозначениями элементов. Так, все объекты, обозначающие вещи, обозначаются в виде прямоугольника. Атрибуты, характеризующие объект - в виде овала, а связи между объектами - ромбами. Мощность связи обозначаются стрелками (в направлении, где мощность равна многим - двойная стрелка, а со стороны, где она равна единице - одинарная).

Этапы концептуального проектирования:

- Определение сущностей

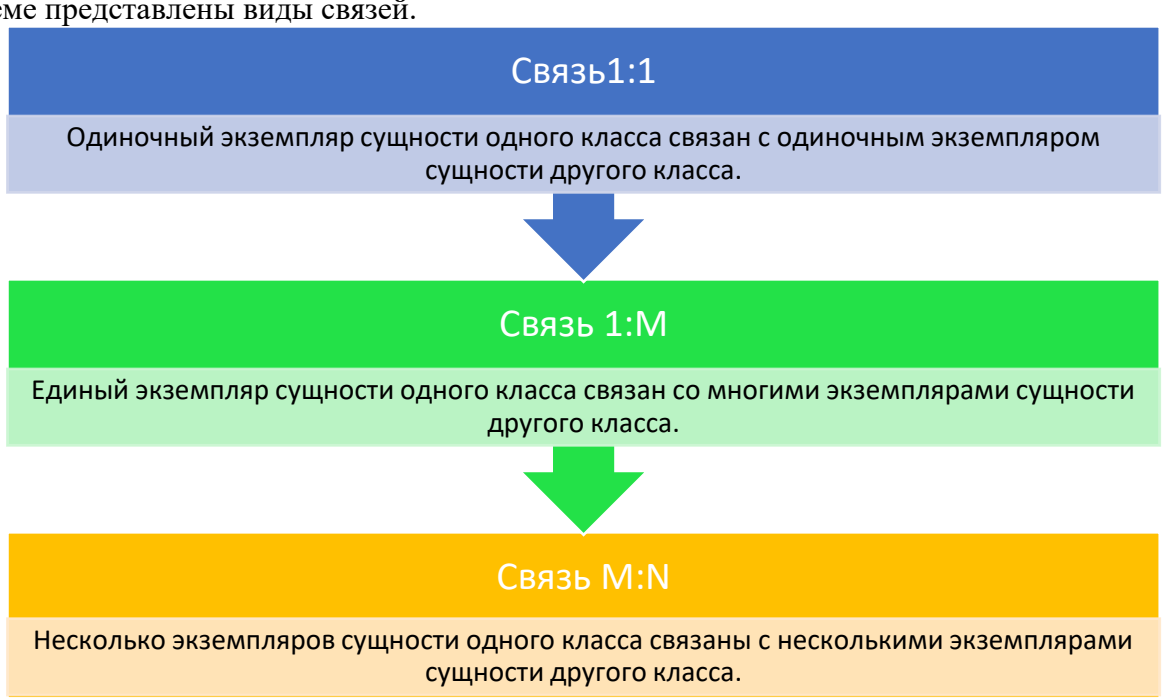
- Определение связей между сущностями
- Создание ER-модели предметной области
- Определение атрибутов (имя, тип, значение по умолчанию ,NULL)
- Определение наборов допустимых значений атрибутов
- Определение первичных ключей для сущностей

ER-диаграммы помогают пользователям моделировать базы данных посредством таблиц, которые обеспечивают им упорядоченность, эффективность и высокую скорость работы. Ну а ключи применяются с целью максимально эффективно связать между собой разные таблицы в базе данных.

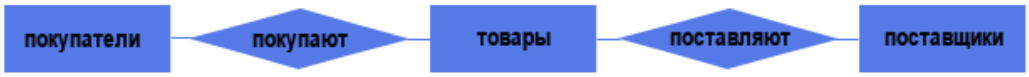
Первичный ключ — это атрибут или сочетание атрибутов, идентифицирующих один конкретный экземпляр сущности.

Внешний ключ создается каждый раз, когда атрибут привязывается к сущности посредством единичной или множественной связи.

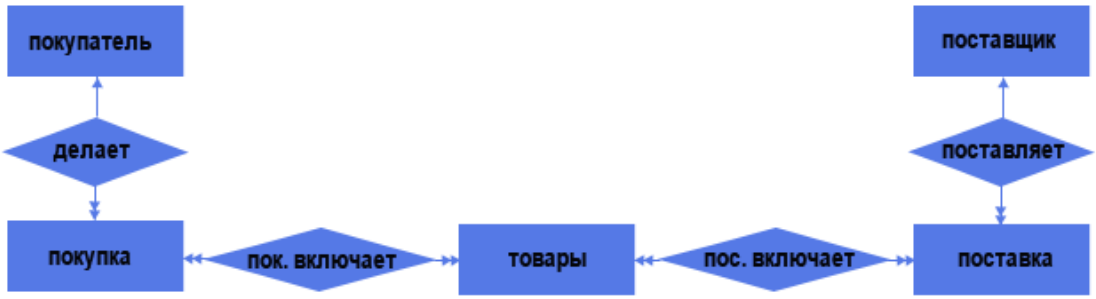
На схеме представлены виды связей.



В качестве примера рассмотрим интернет-магазин. У магазина есть товары, которые поставляются поставщиками и покупаются покупатели. Это можно представить тремя объектами и двумя связями:

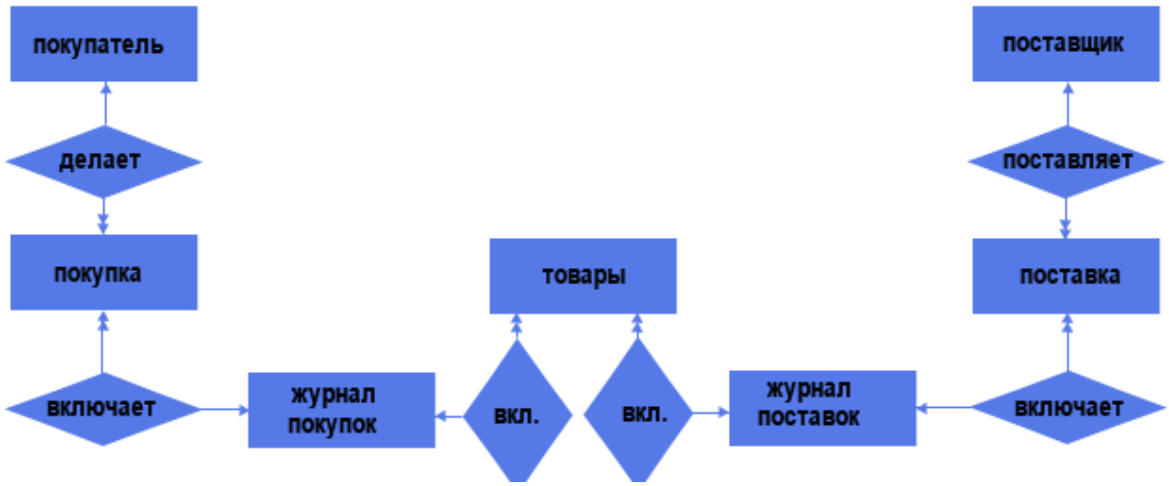


Но как поставщик поставляет товары? Он делает поставку, которая подтверждается документом. Аналогично и покупатель делает покупку, которая также может подтверждаться документом. Таким образом, поставка и покупка могут рассматриваться, как самостоятельные объекты:



Теперь у нас пять объектов и четыре связи. Две связи "один ко многим" (один поставщик может осуществить несколько поставок, но каждая поставка осуществляется только одним поставщиком, аналогично и для связи Покупатель - Покупка) и две связи "многие ко многим" (каждая поставка может содержать несколько товаров, а один и тот же товар может содержаться в нескольких поставках, аналогично и для связи Покупка - Товар).

Но связи "многие ко многим" недопустимы в реляционной модели, поэтому каждую такую связь надо заменить на две связи "один ко многим". Делается это добавлением промежуточного объекта:



Таким образом, у нас появилось еще два объекта - журнал покупок и журнал поставок, со связями "один ко многим" (один журнал поставок может включать несколько поставок, но каждая поставка может входить только в один журнал, аналогично и для остальных).

Каждый объект нашего магазина имеет свои атрибуты:

Таким образом, мы создали концептуальную модель базы данных магазин, вернее ее части, ведь в магазине еще есть сотрудники, склады, доставка товаров и т.д.

Вообще, если предметная область обширная, то ее полезно разбить на несколько локальных предметных областей (наша концептуальная модель отражает именно локальную предметную область). Объем локальной области выбирается таким образом, чтобы в нее входило не более 6-7 объектов. После создания моделей каждой выделенной предметной области производится объединение локальных концептуальных моделей в одну общую, как правило, довольно сложную схему.

Для построения моделей можно пользоваться онлайн средством Lucidchart по ссылке https://www.lucidchart.com/pages/ru?_gl=1*6rkug1*_ga*MTgzNTk0MTI4Mi4xNzAxNTE5NzIx*_ga_MPV5H3XMB5*MTcwMTUxOTcyMS4xLjEuMTcwMTUyMDMyNC40My4wLjA.*_gcl_au*MTEwNTg2NjEyOC4xNzAxNTE5NzIz

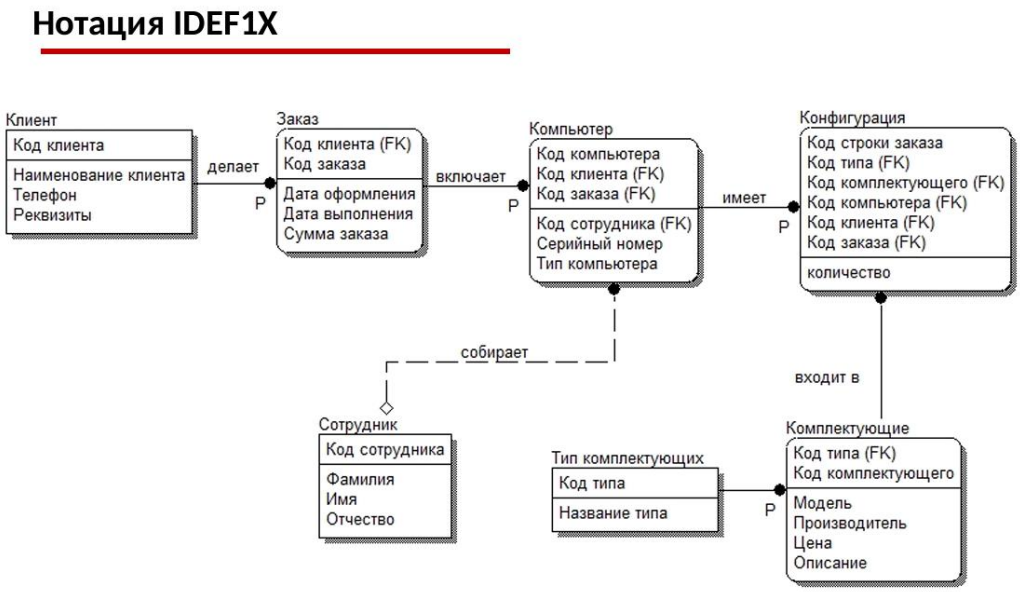
2 Построение логической модели данных

Прикладные программы работают с логической моделью.

Этапы логического моделирования:

- Выбор модели данных
- Определение наборов структурированных данных.
- Нормализация таблиц (для реляционной модели)
- Проверка модели данных на предмет выполнения всех транзакций бизнес-процессов
- Определение требований поддержки целостности данных

Далее представлен пример логической модели в нотации IDEF1X.



3 Физическое проектирование

Этапы логического моделирования:

- Проектирование таблиц средствами выбранной СУБД
- Реализация бизнес-правил в среде выбранной СУБД
- Проектирование физической организации СУБД
- Разработка стратегии защиты БД
- Организация мониторинга функционирования БД и ее настройка

Описать отношения в виде таблиц по форме:
Отношение «Покупатель»

Поле	Тип данных	Описание	Примечания
id_покупателя	int	Номер покупателя	Первичный ключ
f_name	varchar(50)	Фамилия	Not null
name	varchar(50)	Имя	Not null
o_name	varchar(50)	Отчество	Not null
email	varchar(50)	Электронная почта покупателя	Not null

Практическое занятие № 12 Создание серверного приложения преобразованием проекта базы данных формата MS Access в формат Sql-Server

Цель занятия: формирование навыка преобразования проекта базы данных формата MS Access в формат Sql-Server.

Этапы выполнения работы:

1. Выполнить преобразование базы данных Access в формат базы данных MS SQL Server.
2. Выполнить конспект по вопросу «Создание пользовательских представлений».
3. Продемонстрировать полученные результаты преподавателю, защитить выполненную работу.

Литература: Фуфаев, Д.Э., Фуфаев, Э.В. Разработка и эксплуатация автоматизированных информационных систем: учебник пособие для студентов СПО. М.: Академия, 2017. С.150-153, 158-159.

Практическое занятие № 13 Хранимые процедуры

Часть 1

Цель занятия: формирование навыка работы создания процедур

Этапы выполнения работы:

1. Запустить MySql.
2. Выбрать БД magazine (см. Приложение Б).
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');
```

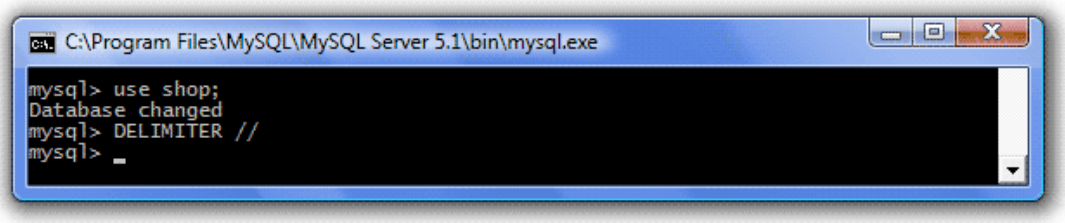
Этот же запрос вполне уместно оформить в виде процедуры:

```
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.

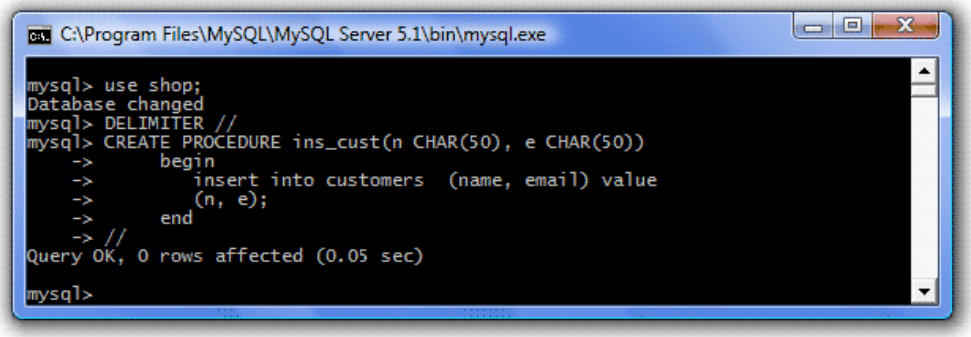
Один нюанс. Как вы помните, точка с запятой означает конец запроса и отправляет его на выполнение, что в данном случае неприемлемо. Поэтому, прежде, чем написать процедуру необходимо переопределить разделитель с ; на "//", чтобы запрос не отправлялся раньше времени. Делается это с помощью оператора DELIMITER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

```
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:

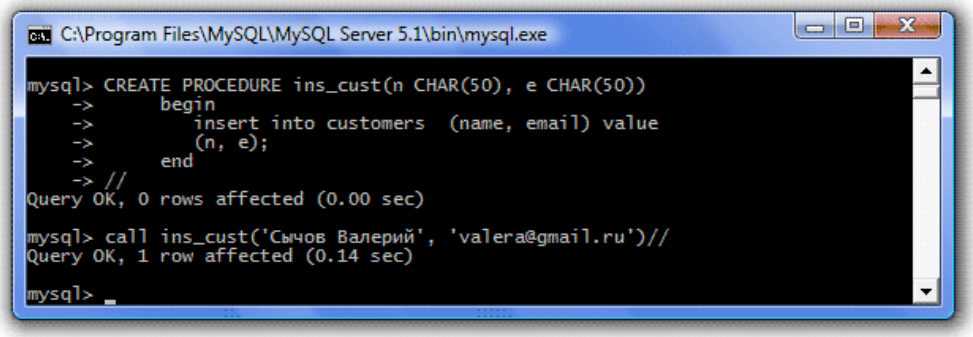
```
CREATE PROCEDURE ins_cust(n CHAR(50), e CHAR(50))
     begin
        insert into customers  (name, email) value (n, e);
     end
//
```



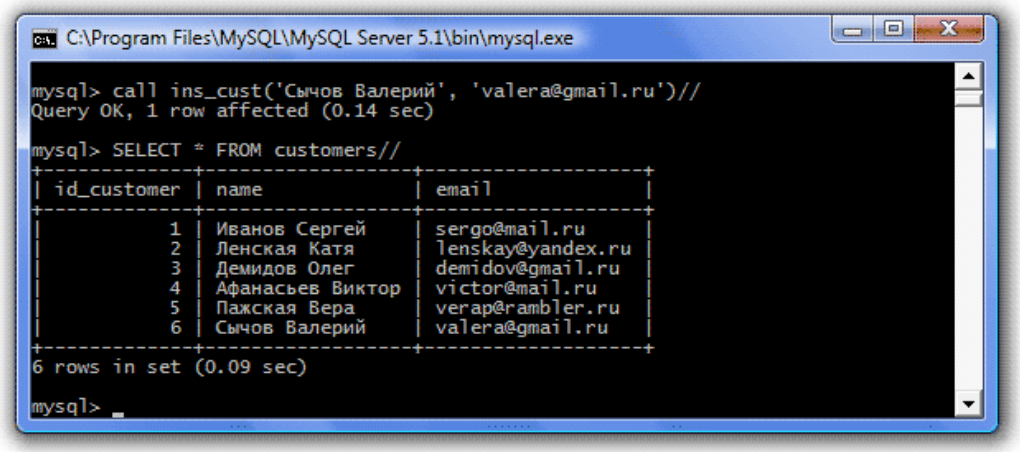
Итак, процедура создана. Теперь, когда нам понадобится ввести нового покупателя нам достаточно ее вызвать, указав необходимые параметры. Для вызова хранимой процедуры используется оператор CALL, после которого указывается имя процедуры и ее параметры.

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):

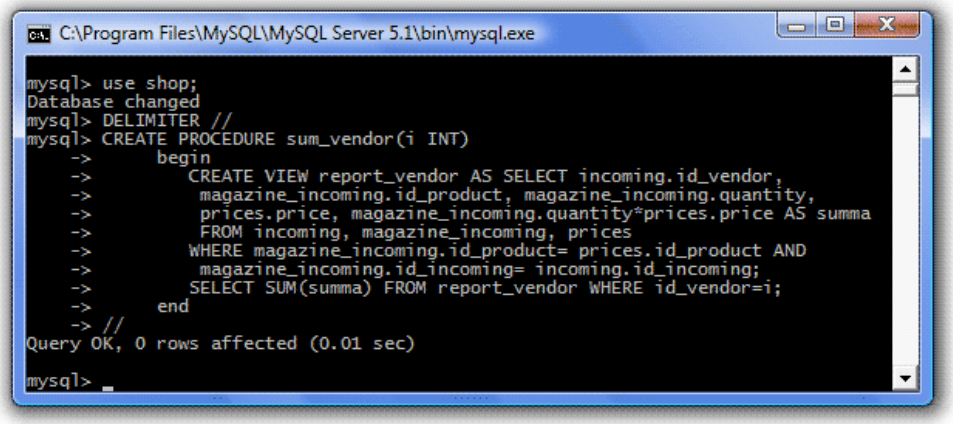
```
CREATE VIEW report_vendor AS SELECT incoming.id_vendor,
  magazine_incoming.id_product, magazine_incoming.quantity,
  prices.price, magazine_incoming.quantity*prices.price AS summa
FROM incoming, magazine_incoming, prices
WHERE magazine_incoming.id_product= prices.id_product AND
magazine_incoming.id_incoming=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

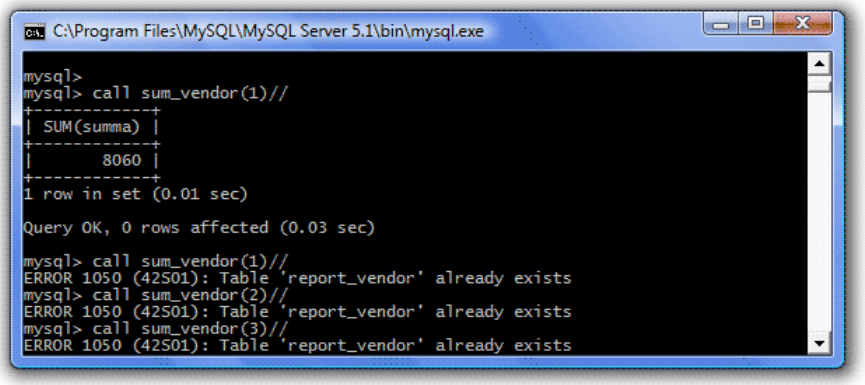
Теперь объедините два этих запроса в хранимую процедуру, где входным параметром будет идентификатор поставщика (id_vendor), который будет подставляться во второй запрос, но не в представление:

```
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)//
```

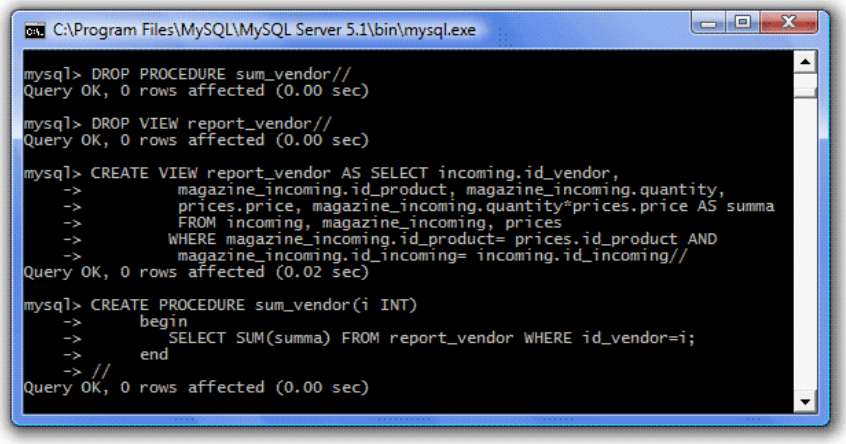


Процедура срабатывает один раз, а затем выдает ошибку, говоря, что представление report_vendor уже имеется в БД.

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.

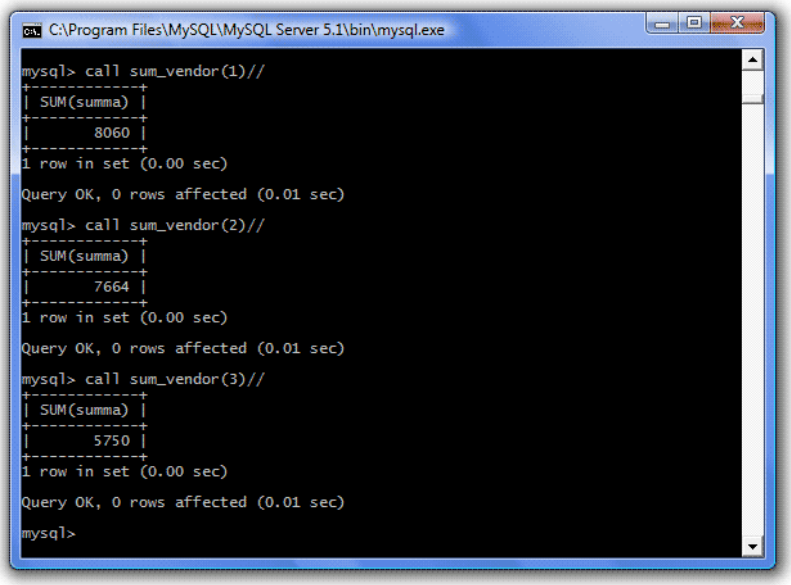
Первый - вынести представление из процедуры. То есть один раз создать представление, а процедура будет лишь к нему обращаться, но не создавать его. Предварительно не забудет удалить уже созданную процедуру и представление:

```
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:

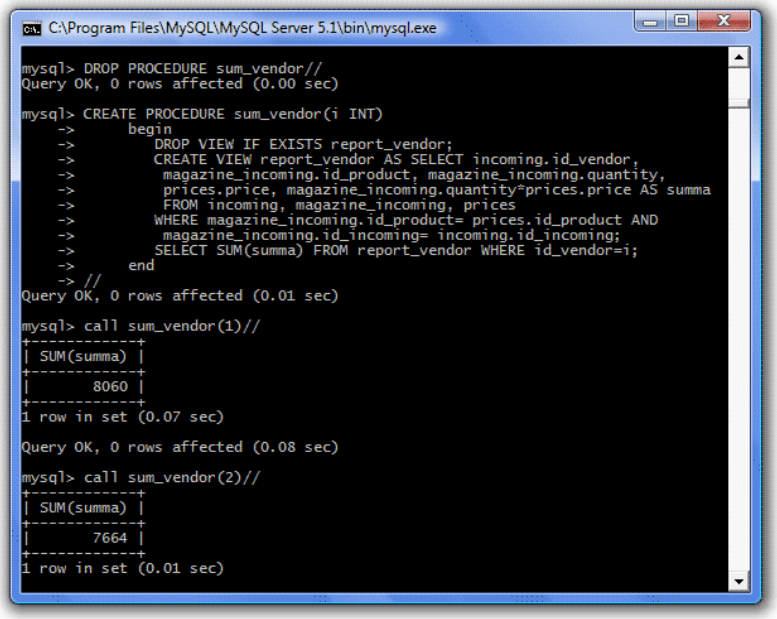
```
call sum_vendor(1)//
call sum_vendor(2)//
call sum_vendor(3)//
```



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:

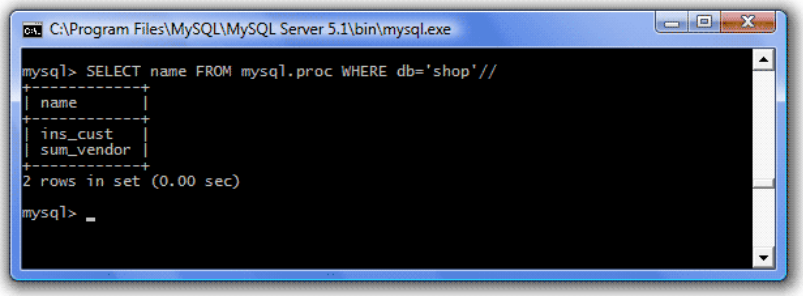
```
SELECT * FROM mysql.proc//
```

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:

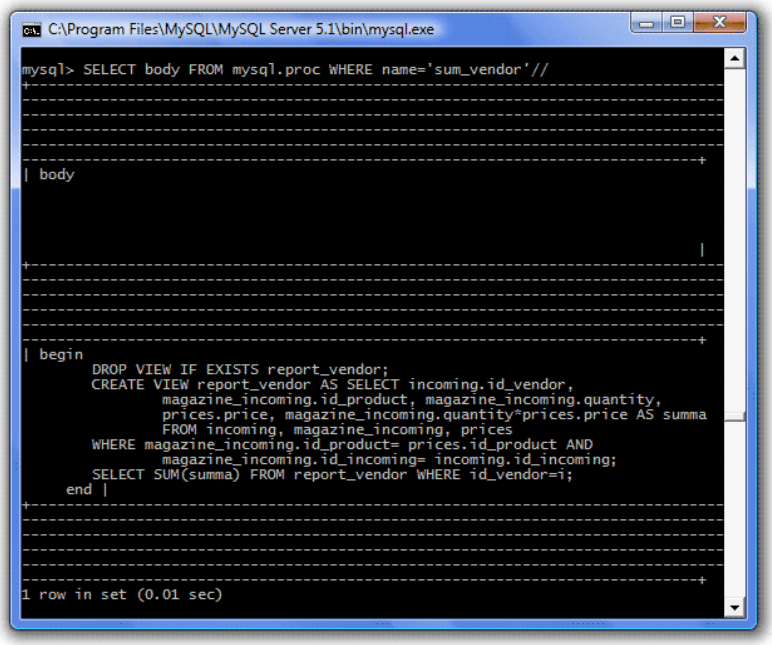
```
SELECT name FROM mysql.proc WHERE db='shop'//
```



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:



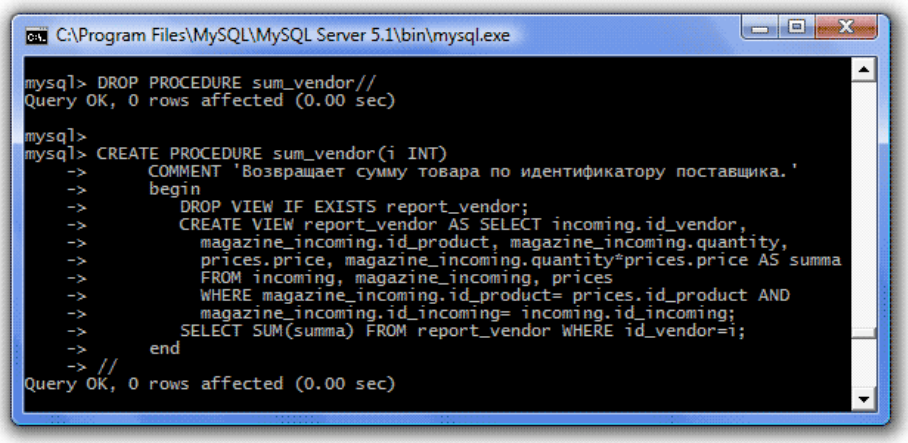
Чтобы извлекать из таблицы proc необходимую информацию, надо просто знать, какие столбцы она содержит, а для этого можно воспользоваться оператором describe имя_таблицы, в данном случае describe mysql.proc.

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.

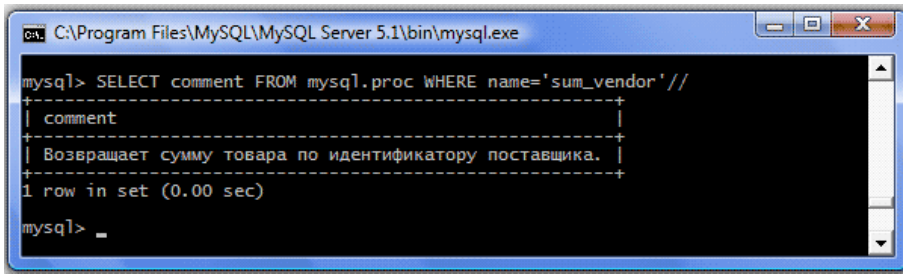
Чтобы создать комментарий, укажите ключевое слово COMMENT 'здесь комментарий'.

```
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'//
```

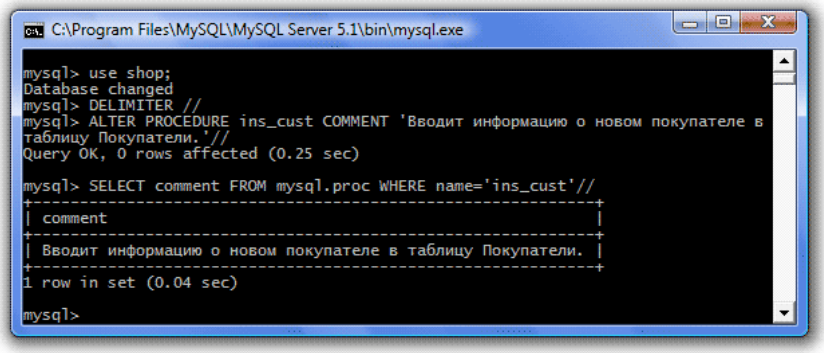


Можно отредактировать имеющуюся хранимую процедуру с помощью оператора ALTER PROCEDURE.

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

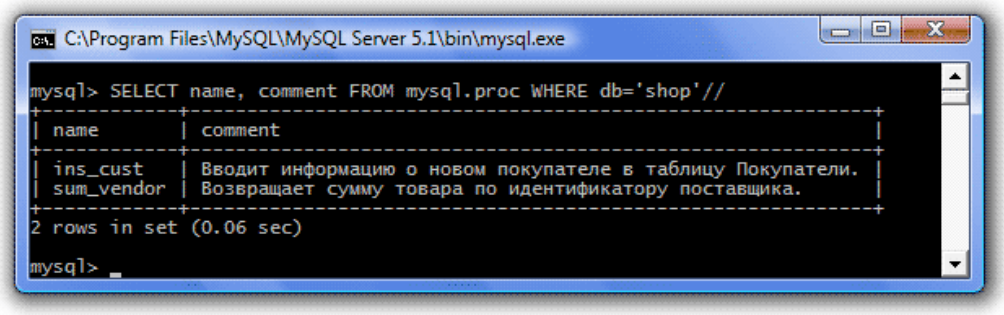
И сделаем запрос к комментарию, чтобы проверить:

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



Запрос для просмотра комментариев:

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



Практическое занятие № 13 Хранимые процедуры

Часть 2

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

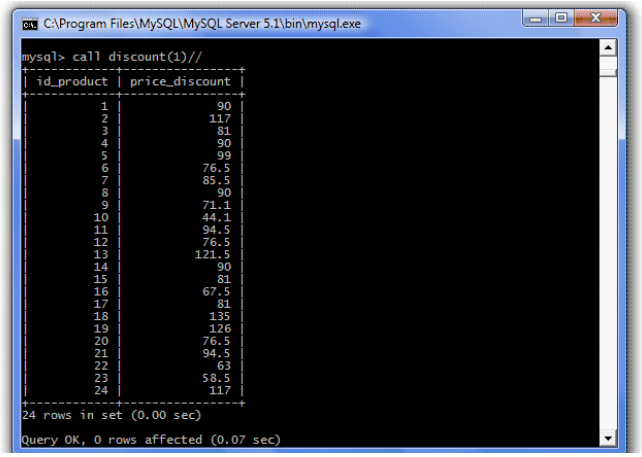
Предположим, каждый день мы устраиваем в нашем магазине счастливые часы, т.е. делаем скидку 10% на все книги в последний час работы магазина. Чтобы иметь возможность выбирать цену книги, нам необходимо иметь два ее варианта - со скидкой и без. Для этого, нам понадобится создать хранимую процедуру с оператором 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
```

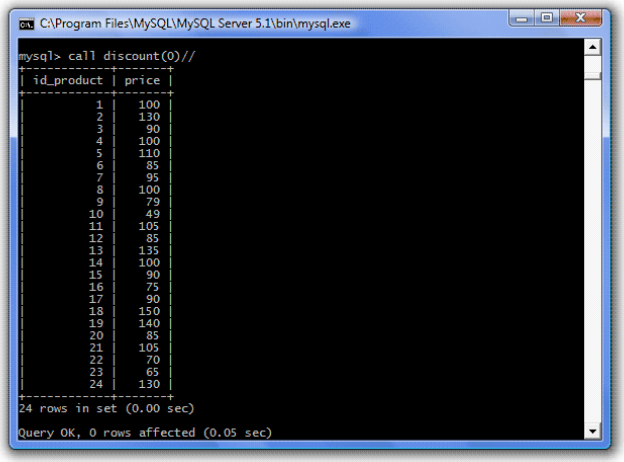
```
        SELECT id_product, price FROM prices;
    END IF;
  end
      //
```

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.

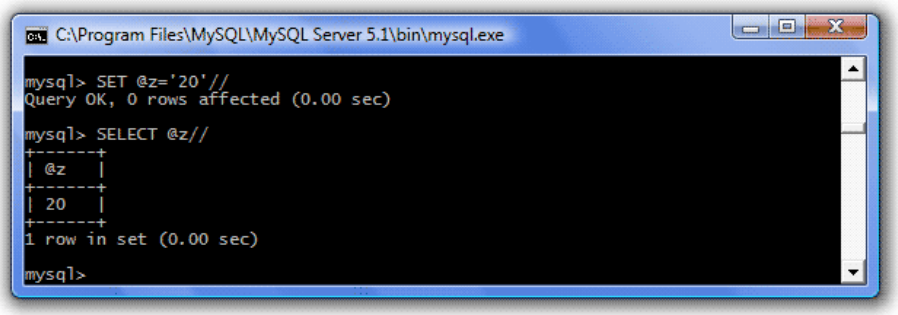
Чтобы передать в процедуру sum_discount результат работы предыдущих запросов. введем переменную.

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.

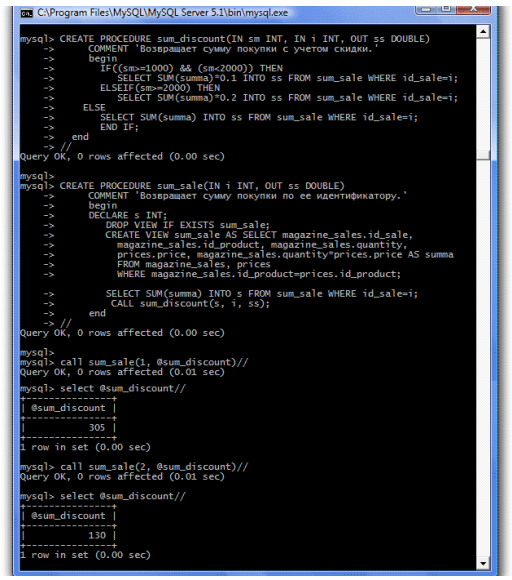
– Затем выполняется запрос к этому представлению на итоговую сумму по покупке с нужным идентификатором, и результат записывается в переменную s.

– 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.

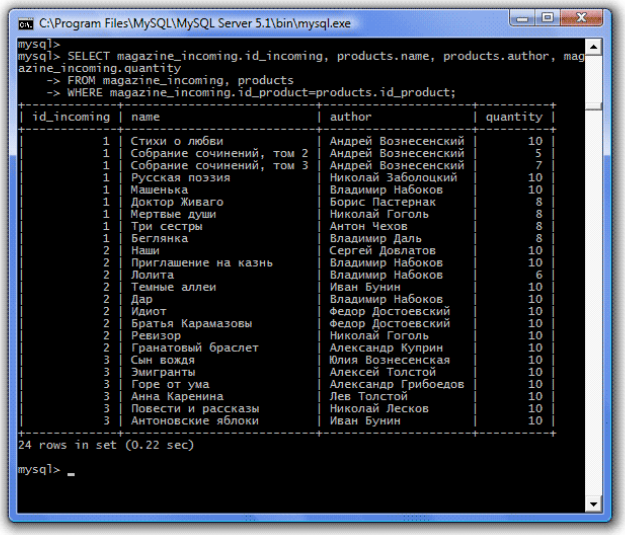
Оператор цикла REPEAT	`REPEAT`
	`запрос`
	`UNTIL условие`
	`END REPEAT`
		Условие цикла проверяется не в начале, как в цикле WHILE, а в конце, т.е. хотя бы один раз, но цикл выполняется. Сам же цикл выполняется, пока условие ложно

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:

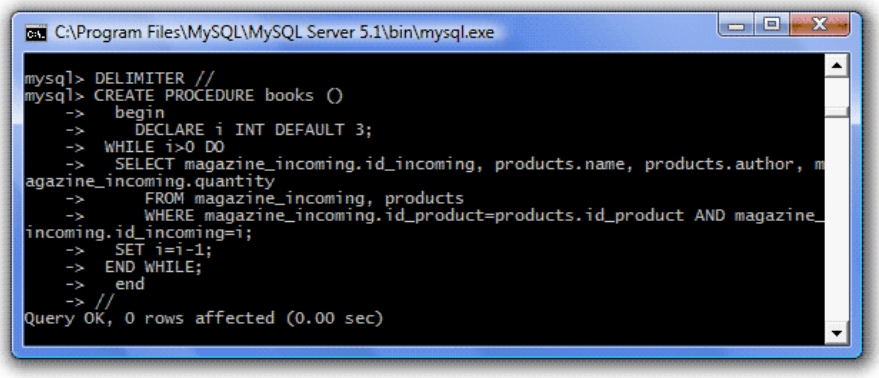
```
DECLARE i INT DEFAULT 3;
   WHILE i>0 DO
     SELECT magazine_incoming.id_incoming, products.name, products.author,
magazine_incoming.quantity
     FROM magazine_incoming, products
     WHERE magazine_incoming.id_product=products.id_product AND
magazine_incoming.id_incoming=i;
     SET i=i-1;
  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

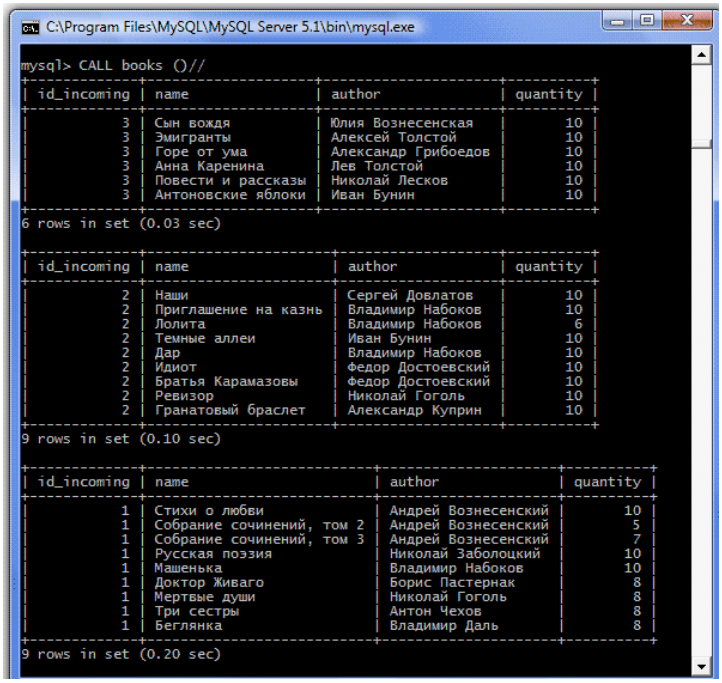
Чтобы убедиться в работоспособности цикла создадим хранимую процедуру books и поместим в нее цикл:

```
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:

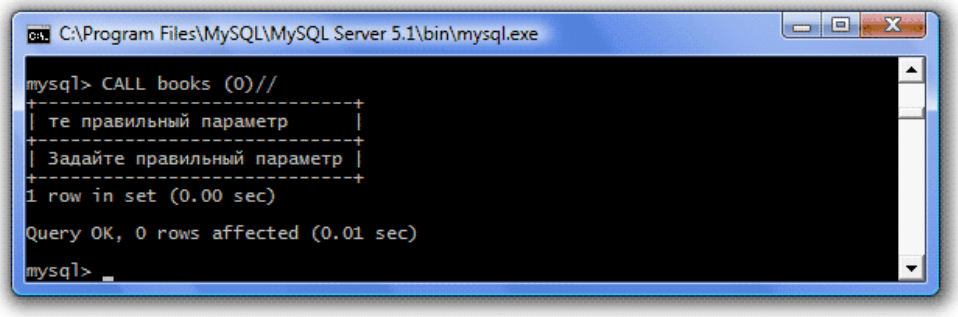
```
CALL books ()//
```



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 № 14 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.

Приложение А

Список тем для разработки индивидуальной информационной системы

1. Автозвук
2. Автосалон
3. Автосервис
4. Агентство недвижимости
5. Отдел кредитования в банке
6. Выставка картин
7. Грузоперевозки
8. Заказ билетов
9. Магазин компьютерной техники
10. Магазин одежды
11. Магазин сантехники
12. Магазин цветов
13. Микрокредитование
14. Пиццерия
15. Продажа картин
16. Рекламное агентство
17. Ремонт компьютерной техники
18. Ремонт и пошив одежды
19. Репетитор
20. Салон красоты
21. Свадебный салон
22. Система охраны
23. Страйкбольный магазин
24. Строительная компания
25. Строительный магазин
26. Туристическое агентство
27. Фирма по доставке питьевой воды
28. Фотоателье
29. Шиномонтаж
30. Школа по изучению иностранных языков

Приложение Б

Создаем в БД Магазин 8 таблиц, как в схеме:

- Покупатели (customers),
- Поставщики (vendors),
- Покупки (sale),
- Поставки (incoming),
- Журнал покупок (magazine_sales),
- Журнал поставок (magazine_incoming),
- Товары (products),
- Цены (prices).

Один нюанс, магазин будет торговать книгами, поэтому в таблицу Товары мы добавим еще один столбец - Автор (author).

```
create table customers (
  id_customer int NOT NULL AUTO_INCREMENT,
  name char(50) NOT NULL,
  email char(50) NOT NULL,
  PRIMARY KEY (id_customer)
);

create table vendors (
  id_vendor int NOT NULL AUTO_INCREMENT,
  name char(50) NOT NULL,
  city char(30) NOT NULL,
  address char(100) NOT NULL,
  PRIMARY KEY (id_vendor)
);

create table sale (
  id_sale int NOT NULL AUTO_INCREMENT,
  id_customer int NOT NULL,
  date_sale date NOT NULL,
  PRIMARY KEY (id_sale),
  FOREIGN KEY (id_customer) REFERENCES customers (id_customer)
);

create table incoming (
  id_incoming int NOT NULL AUTO_INCREMENT,
  id_vendor int NOT NULL,
  date_incoming date NOT NULL,
  PRIMARY KEY (id_incoming),
  FOREIGN KEY (id_vendor) REFERENCES vendors (id_vendor)
);

create table products (
  id_product int NOT NULL AUTO_INCREMENT,
  name char(100) NOT NULL,
  author  char(50) NOT NULL,
  PRIMARY KEY (id_product)
);

create table prices (
  id_product int NOT NULL,
  date price changes date NOT NULL,
  price double NOT NULL,
```

```
  PRIMARY KEY (id_product, date_price_changes),
  FOREIGN KEY (id_product) REFERENCES products (id_product)
);

create table magazine_sales (
  id_sale int NOT NULL,
  id_product int NOT NULL,
  quantity int NOT NULL,
  PRIMARY KEY (id_sale, id_product),
  FOREIGN KEY (id_sale) REFERENCES sale (id_sale),
  FOREIGN KEY (id_product) REFERENCES products (id_product)
);

create table magazine_incoming (
  id_incoming int NOT NULL,
  id_product int NOT NULL,
  quantity int NOT NULL,
  PRIMARY KEY (id_incoming, id_product),
  FOREIGN KEY (id_incoming) REFERENCES incoming (id_incoming),
  FOREIGN KEY (id_product) REFERENCES products (id_product)
);
```

Обратите внимание, что в таблицах Журнал покупок, Журнал поставок и Цены первичные ключи - составные, т.е. их уникальные значения состоят из пар значений (в таблице не может быть двух строк с одинаковыми парами значений). Названия столбцов этих пар значений и указываются через запятую после ключевого слова PRIMARY KEY.

В настоящем интернет-магазине данные в эти таблицы будут заноситься посредством сценариев на каком-либо языке (типа php), нам же пока придется внести их вручную. Можете внести любые данные, только помните, что значения в одноименных столбцах связанных таблиц должны совпадать. Либо скопируйте нижеприведенные данные:

```
INSERT INTO vendors  (name, city, address) VALUES
 ('Вильямс', 'Москва', 'ул.Лесная, д.43'),
 ('Дом печати', 'Минск', 'пр.Ф.Скорины, д.18'),
 ('БХВ-Петербург', 'Санкт-Петербург', 'ул.Есенина, д.5');

INSERT INTO customers  (name, email) VALUES
 ('Иванов Сергей', 'sergo@mail.ru'),
 ('Ленская Катя', 'lenskay@yandex.ru'),
 ('Демидов Олег', 'demidov@gmail.ru'),
 ('Афанасьев Виктор', 'victor@mail.ru'),
 ('Пажская Вера', 'verap@rambler.ru');

INSERT INTO products  (name, author) VALUES
 ('Стихи о любви', 'Андрей Вознесенский'),
 ('Собрание сочинений, том 2', 'Андрей Вознесенский'),
 ('Собрание сочинений, том 3', 'Андрей Вознесенский'),
 ('Русская поэзия', 'Николай Заболоцкий'),
 ('Машенька', 'Владимир Набоков'),
 ('Доктор Живаго', 'Борис Пастернак'),
 ('Наши', 'Сергей Довлатов'),
 ('Приглашение на казнь', 'Владимир Набоков'),
 ('Лолита', 'Владимир Набоков'),
 ('Темные аллеи', 'Иван Бунин'),
 ('Дар', 'Владимир Набоков'),
 ('Сын вождя', 'Юлия Вознесенская'),
 ('Эмигранты', 'Алексей Толстой'),
 ('Горе от ума', 'Александр Грибоедов'),
 ('Анна Каренина', 'Лев Толстой'),
 ('Повести и рассказы', 'Николай Лесков'),
 ('Антоновские яблоки', 'Иван Бунин'),
 ('Мертвые души', 'Николай Гоголь'),
```

```
 ('Три сестры', 'Антон Чехов'),
 ('Беглянка', 'Владимир Даль'),
 ('Идиот', 'Федор Достоевский'),
 ('Братья Карамазовы', 'Федор Достоевский'),
 ('Ревизор', 'Николай Гоголь'),
 ('Гранатовый браслет', 'Александр Куприн');

INSERT INTO incoming  (id_vendor, date_incoming) VALUES
 ('1', '2023-04-10'),
 ('2', '2023-04-11'),
 ('3', '2023-04-12');

INSERT INTO magazine_incoming  (id_incoming, id_product, quantity) VALUES
 ('1', '1', '10'),
 ('1', '2', '5'),
 ('1', '3', '7'),
 ('1', '4', '10'),
 ('1', '5', '10'),
 ('1', '6', '8'),
 ('1', '18', '8'),
 ('1', '19', '8'),
 ('1', '20', '8'),
 ('2', '7', '10'),
 ('2', '8', '10'),
 ('2', '9', '6'),
 ('2', '10', '10'),
 ('2', '11', '10'),
 ('2', '21', '10'),
 ('2', '22', '10'),
 ('2', '23', '10'),
 ('2', '24', '10'),
 ('3', '12', '10'),
 ('3', '13', '10'),
 ('3', '14', '10'),
 ('3', '15', '10'),
 ('3', '16', '10'),
 ('3', '17', '10');

INSERT INTO prices  (id_product, date_price_changes, price) VALUES
 ('1', '2023-04-10', '100'),
 ('2', '2023-04-10', '130'),
 ('3', '2023-04-10', '90'),
 ('4', '2023-04-10', '100'),
 ('5', '2023-04-10', '110'),
 ('6', '2023-04-10', '85'),
 ('7', '2023-04-11', '95'),
 ('8', '2023-04-11', '100'),
 ('9', '2023-04-11', '79'),
 ('10', '2023-04-11', '49'),
 ('11', '2023-04-11', '105'),
 ('12', '2023-04-12', '85'),
 ('13', '2023-04-12', '135'),
 ('14', '2023-04-12', '100'),
 ('15', '2023-04-12', '90'),
 ('16', '2023-04-12', '75'),
 ('17', '2023-04-12', '90'),
 ('18', '2023-04-10', '150'),
 ('19', '2023-04-10', '140'),
 ('20', '2023-04-10', '85'),
 ('21', '2023-04-11', '105'),
 ('22', '2023-04-11', '70'),
 ('23', '2023-04-11', '65'),
 ('24', '2023-04-11', '130');
```

```
INSERT INTO sale  (id_customer, date_sale) VALUES
 ('2', '2023-04-11'),
 ('3', '2023-04-11'),
 ('5', '2023-04-11');

INSERT INTO magazine_sales  (id_sale, id_product, quantity) VALUES
 ('1', '1', '1'),
 ('1', '5', '1'),
 ('1', '7', '1'),
 ('2', '2', '1'),
 ('3', '1', '1'),
 ('3', '7', '1');
```