Задания на лабораторную работу:

- изучить теоретический материал;
- выполнить задания.

Теоретический материал

Команда CREATE DATABASE

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. 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. 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.

Пример.

```
create database employees;
```

Работа с таблицами

Команда SQL для *создания таблицы* выглядит следующим образом:

```
CREATE TABLE employee_data
(
emp_id int unsigned not null auto_increment primary key,
f_name varchar(20),
l_name varchar(20),
title varchar(30),
age int,
yos int,
salary int,
perks int,
email varchar(60)
);
```

Примечание: в MySQL команды и имена столбцов не различают регистр символов, однако имена таблиц и баз данных могут зависеть от регистра в связи с используемой платформой (как в Linux). Поэтому можно вместо `CREATE TABLE` использовать `create table`.
За ключевыми словами `CREATE TABLE` следует имя создаваемой таблицы `employee_data`. Каждая строка внутри скобок представляет один столбец. Эти столбцы хранят для каждого сотрудника идентификационный номер (`emp_id`), фамилию (`f_name`), имя

(l_name), должность (title), возраст (age), стаж работы в компании (yos), зарплату (salary), надбавки (perks), и адрес e-mail (email).
За именем каждого столбца следует тип столбца. Типы столбцов определяют тип данных, которые будет содержать столбец. В данном примере столбцы f_name, l_name, title и email будут содержать текстовые строки, поэтому тип столбца задан как varchar, что означает переменное количество символов. Максимальное число символов для столбцов varchar определяется числом, заключенным в скобки, которое следует сразу за именем столбца. Столбцы age, yos, salary и perks будут содержать числа (целые), поэтому тип столбца задается как int. Первый столбец (emp_id) содержит идентификационный номер (id) сотрудника. Его тип столбца выглядит несколько перегруженным, поэтому рассмотрим его по частям:

- 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.

Использование базы данных

База данных employees уже создана. Для работы с ней, необходимо её "активировать" или "выбрать". В приглашении mysql выполните команду:

```
SELECT DATABASE();
```

На экране увидим ответ системы

Это говорит о том, что ни одна база данных не была выбрана. На самом деле всякий раз при работе с клиентом mysql необходимо определять, какая *база данных будет использоваться*.

Определить текущую базу данных можно несколькими способами:

- определение имени базы данных при запуске

Введите в приглашении системы следующее:

```
mysql employees (в Windows)
mysql employees -u manish -p (в Linux)
```

- определение базы данных с помощью оператора USE в приглашении mysql

```
mysql>USE employees;
```

- Определение базы данных с помощью `\u` в приглашении `mysql`

```
mysql>\u employees;
```

При работе необходимо определять *базу данных, которая будет использоваться*, иначе MySQL будет порождать ошибку.

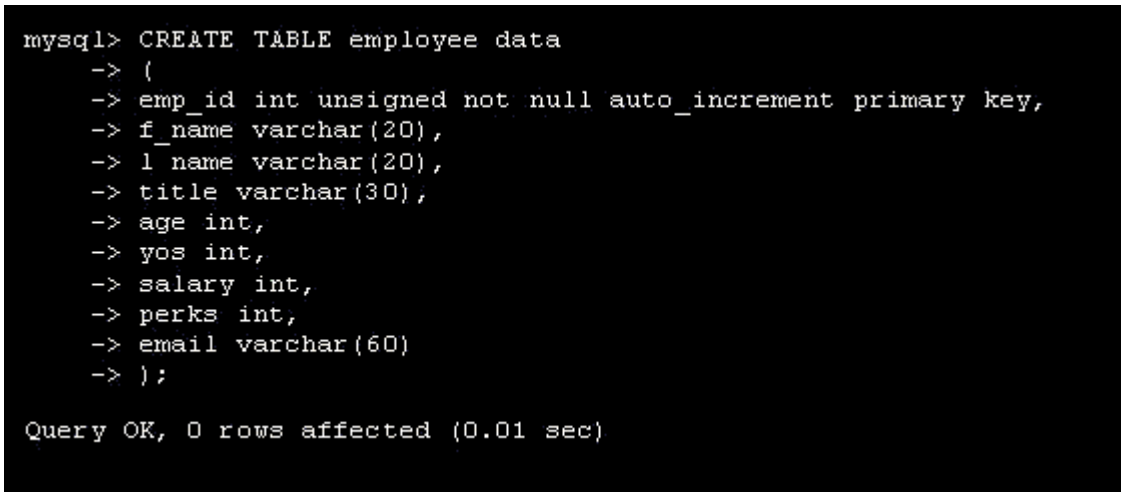
Создание таблицы

После выбора базы данных `employees`, выполните в приглашении `mysql` команду `CREATE TABLE`.

```
CREATE TABLE employee_data
(
emp_id int unsigned not null auto_increment primary key,
f_name varchar(20),
l_name varchar(20),
title varchar(30),
age int,
yos int,
salary int,
perks int,
email varchar(60)
);
```

Нажатие клавиши Enter после ввода первой строки изменяет приглашение `mysql` на `->`. Это означает, что `mysql` понимает, что команда не завершена и приглашает ввести дополнительные операторы. Помните, что каждая команда `mysql` заканчивается точкой с запятой, а каждое объявление столбца отделяется запятой. Можно также при желании ввести всю команду на одной строке.

Вывод на экране должен соответствовать рисунку



Синтаксис команды CREATE TABLE

Общий формат инструкции `CREATE TABLE` таков:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] имя
[(спецификация, ...)]
[опция, ...]
[ [IGNORE | REPLACE] запрос]
```

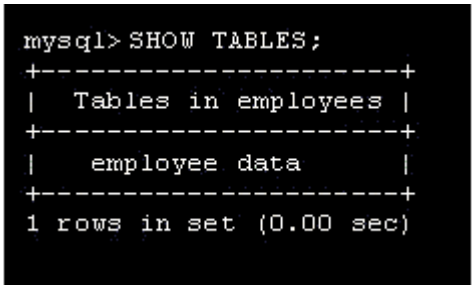
Флаг TEMPORARY задает *создание временной таблицы*, существующей в течение текущего сеанса. По завершении сеанса таблица удаляется. Временным таблицам можно присваивать имена других таблиц, делая последние временно недоступными. Спецификатор IF NOT EXIST подавляет вывод сообщений об ошибках в случае, 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.
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]
```

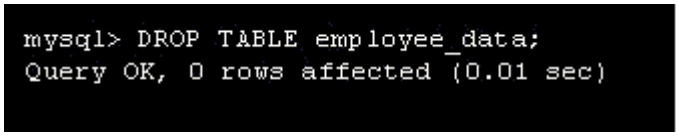
Спецификация типа включает название типа и его размерность. По умолчанию столбцы принимают значения NULL. Спецификатор NOT NULL запрещает подобное поведение.
У любого столбца есть значение по умолчанию. Если оно не указано, программа MySQL выберет его самостоятельно. Для столбцов, принимающих значения 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.

Удаление таблиц

Для того, чтобы удалить таблицу, убедимся сперва что она существует. Это можно проверить с помощью команды SHOW TABLES, как показано на рисунке



Для *удаления таблицы* используется команда DROP TABLE, как показано на рисунке



Теперь команда SHOW TABLES; этой таблицы больше не покажет.

Синтаксис команды DROP TABLE

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.

Типы данных столбцов

MySQL поддерживает несколько *типов столбцов*, которые можно разделить на три категории: *числовые типы* данных, типы данных для хранения даты и времени и символьные (строковые) типы данных. В данной лекции мы вначале рассмотрим все возможные типы и приведём требования по хранению для каждого типа столбца, затем опишем свойства типов более подробно по каждой категории.

Ниже перечислены *типы столбцов*, поддерживаемые MySQL. В описаниях используются обозначения, которые использовали разработчики MySql в официальной документации:

- `M` - указывает максимальный размер вывода. Максимально допустимый размер вывода составляет 255 символов.
- `D` - употребляется для типов данных с плавающей точкой и указывает количество разрядов, следующих за десятичной точкой. Максимально возможная величина составляет 30 разрядов, но не может быть больше, чем M-2.

Квадратные скобки ('[' и ']') указывают для типа данных группы необязательных признаков.

Типы полей MySql.

Таблица 4.1. Типы полей MySql

`TINYINT[(M)] [UNSIGNED] [ZEROFILL]`	Очень малое целое число. Диапазон со знаком от -128 до 127. Диапазон без знака от 0 до 255
`BIT, BOOL`	Синонимы `TINYINT(1)`
`SMALLINT[(M)] [UNSIGNED] [ZEROFILL]`	Малое целое число. Диапазон со знаком от -32768 до 32767. Диапазон без знака от 0 до 65535.
`MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]`	Целое число среднего размера. Диапазон со знаком от -8388608 до 8388607. Диапазон без знака от 0 до 16777215
`INT[(M)] [UNSIGNED] [ZEROFILL]`	Целое число нормального размера. Диапазон со знаком от -2147483648 до 2147483647. Диапазон без знака от 0 до 4294967295.
`INTEGER[(M)] [UNSIGNED] [ZEROFILL]`	Синоним для `INT`
`BIGINT[(M)] [UNSIGNED] [ZEROFILL]`	Большое целое число. Диапазон со знаком от -9223372036854775808 до 9223372036854775807. Диапазон без знака от 0 до 18446744073709551615
`FLOAT(точность) [UNSIGNED] [ZEROFILL]`	Число с плавающей точкой. Атрибут точности может иметь значение <=24 для числа с плавающей точкой обычной (одинарной) точности и между 25 и 53 - для числа с плавающей точкой удвоенной точности. Эти типы данных сходны с типами `FLOAT` и `DOUBLE`, описанными ниже. `FLOAT(X)` относится к тому же интервалу, что и соответствующие типы `FLOAT` и `DOUBLE`, но диапазон значений и количество

	десятичных знаков не определены.
FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]	Малое число с плавающей точкой обычной точности. Допустимые значения: от -3,402823466E+38 до -1,175494351E-38, 0, и от 1,175494351E-38 до 3,402823466E+38. Если указан атрибут UNSIGNED, отрицательные значения недопустимы. Атрибут M указывает количество выводимых пользователю знаков, а атрибут D - количество разрядов, следующих за десятичной точкой. Обозначение FLOAT без указания аргументов или запись вида FLOAT(X), где X <=24 справедливы для числа с плавающей точкой обычной точности.
DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]	Число с плавающей точкой удвоенной точности нормального размера. Допустимые значения: от -1,7976931348623157E+308 до -2,2250738585072014E-308, 0, и от 2,2250738585072014E-308 до 1,7976931348623157E+308. Если указан атрибут UNSIGNED, отрицательные значения недопустимы. Атрибут M указывает количество выводимых пользователю знаков, а атрибут D - количество разрядов, следующих за десятичной точкой. Обозначение DOUBLE без указания аргументов или запись вида FLOAT(X), где 25 <= X <= 53 справедливы для числа с плавающей точкой двойной точности.
DECIMAL[(M[,D])][UNSIGNED] [ZEROFILL] или DEC[(M[,D])] [UNSIGNED] [ZEROFILL] или NUMERIC[(M[,D])][UNSIGNED] [ZEROFILL]	"Неупакованное" число с плавающей точкой. Ведет себя подобно столбцу CHAR, содержащему цифровое значение. Термин "неупакованное" означает, что число хранится в виде строки и при этом для каждого десятичного знака используется один символ. Разделительный знак десятичных разрядов, а также знак '-' для отрицательных чисел не учитываются в M (но место для них зарезервировано). Если атрибут D равен 0, величины будут представлены без десятичного знака, т.е. без дробной части. Максимальный интервал значений типа DECIMAL тот же, что и для типа DOUBLE, но действительный интервал для конкретного столбца DECIMAL может быть ограничен выбором значений атрибутов M и D. Если указан атрибут UNSIGNED, отрицательные значения недопустимы. Если атрибут D не указан, его значение по умолчанию равно 0. Если не указан M, его значение по умолчанию равно 10.
DATE	Дата. Поддерживается интервал от '1000-01-01' до '9999-12-31'. MySQL выводит значения DATE в

	формате 'YYYY-MM-DD', но можно установить значения в столбец DATE, используя как строки, так и числа.
DATETIME	Комбинация даты и времени. Поддерживается интервал от '1000-01-01 00:00:00' до '9999-12-31 23:59:59'. MySQL выводит значения DATETIME в формате 'YYYY-MM-DD HH:MM:SS', но можно устанавливать значения в столбце DATETIME, используя как строки, так и числа.
TIMESTAMP[(M)]	Временная метка. Интервал от '1970-01-01 00:00:00' до некоторого значения времени в 2037 году. MySQL выводит значения TIMESTAMP в форматах YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD или YYMMDD в зависимости от значений M: 14 (или отсутствующее), 12, 8, или 6; но можно также устанавливать значения в столбце TIMESTAMP, используя как строки, так и числа. Столбец TIMESTAMP полезен для записи даты и времени при выполнении операций INSERT или UPDATE, так как при этом автоматически вносятся значения даты и времени самой последней операции, если эти величины не введены программой. Можно также устанавливать текущее значение даты и времени, задавая значение NULL.
TIME	Время. Интервал от '-838:59:59' до '838:59:59'. MySQL выводит значения TIME в формате 'HH:MM:SS', но можно устанавливать значения в столбце TIME, используя как строки, так и числа.
YEAR[(2|4)]	Год в двухзначном или четырехзначном форматах (по умолчанию формат четырехзначный). Допустимы следующие значения: с 1901 по 2155, 0000 для четырехзначного формата года и 1970-2069 при использовании двухзначного формата (70-69). MySQL выводит значения YEAR в формате YYYY, но можно задавать значения в столбце YEAR, используя как строки, так и числа.
[NATIONAL] CHAR(M) [BINARY]	Строка фиксированной длины, при хранении всегда дополняется пробелами в конце строки до заданного размера. Диапазон аргумента M составляет от 0 до 255 символов. Концевые пробелы удаляются при выводе значения. Если не задан атрибут чувствительности к регистру BINARY, то величины CHAR сортируются и сравниваются как независимые от регистра в соответствии с установленным по умолчанию алфавитом. Атрибут NATIONAL CHAR (или его эквивалентная краткая форма NCHAR) представляет собой принятый в ANSI SQL способ

	указания, что в столбце CHAR должен использоваться установленный по умолчанию набор символов (CHARACTER).
CHAR	Это синоним для CHAR(1).
[NATIONAL] VARCHAR(M) [BINARY]	Строка переменной длины. Примечание: концевые пробелы удаляются при сохранении значения (в этом заключается отличие от спецификации ANSI SQL). Диапазон аргумента M составляет от 0 до 255 символов. Если не задан атрибут чувствительности к регистру BINARY, то величины VARCHAR сортируются и сравниваются как независимые от регистра.
TINYBLOB, TINYTEXT	Столбец типа BLOB или TEXT с максимальной длиной 255 символов
BLOB, TEXT	Столбец типа BLOB или TEXT с максимальной длиной 65535 символов.
MEDIUMBLOB, MEDIUMTEXT	Столбец типа BLOB или TEXT с максимальной длиной 16777215 символов.
LONGBLOB, LONGTEXT	Столбец типа BLOB или TEXT с максимальной длиной 4294967295 символов.
ENUM('значение1','значение2',...)	Перечисляемый тип данных. Объект строки может иметь только одно значение, выбранное из заданного списка величин 'значение1', 'значение2', ..., NULL или специальная величина ошибки "". Список ENUM может содержать максимум 65535 различных величин
SET('значение1','значение2',...)	Набор. Объект строки может иметь ноль или более значений, каждое из которых должно быть выбрано из заданного списка величин 'значение1', 'значение2', ... Список SET может содержать максимум 64 элемента.

Запись данных в таблицы

Оператор INSERT заполняет таблицу данными. Вот общая форма INSERT.

```
INSERT into table_name (column1, column2, ...)
values (value1, value2...);
```

где table_name является именем таблицы, в которую надо внести данные; column1, column2 и т.д. являются именами столбцов, а value1, value2 и т.д. являются значениями для соответствующих столбцов.

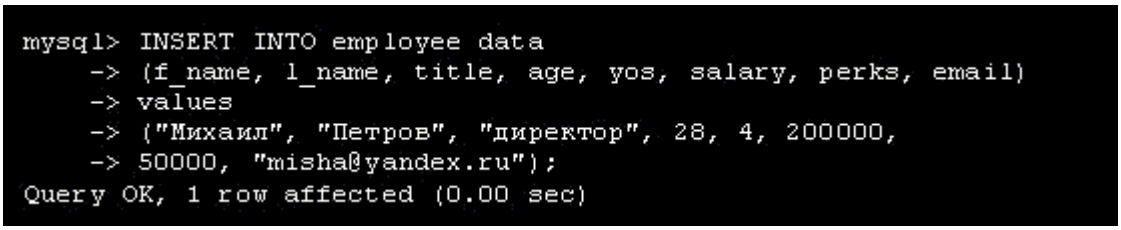
Следующий оператор вносит первую *запись в таблицу* employee_data, которую мы создали ранее:

```
INSERT INTO employee_data
(f_name, l_name, title, age, yos, salary, perks, email)
values
("Михаил", "Петров", "директор", 28, 4, 200000,
50000, "misha@yandex.ru");
```

Как и другие операторы MySQL, эту команду можно вводить на одной строке или разместить ее на нескольких строках.
Несколько важных моментов:

- Значениями для столбцов `f_name`, `l_name`, `title` и `email` являются текстовые строки, и они записываются в кавычках.
- Значениями для `age`, `yos`, `salary` и `perks` являются числа (целые), и они не имеют кавычек.
- Можно видеть, что данные заданы для всех столбцов кроме `except emp_id`. Значение для этого столбца задает система MySQL, которая находит в столбце наибольшее значение, увеличивает его на единицу, и вставляет новое значение.

Если приведенная выше команда правильно введена в приглашении клиента `mysql`, то программа выведет сообщение об успешном выполнении, как показано на рисунке:



Создание дополнительных записей требует использования отдельных операторов `INSERT`.

Запрос данных из таблицы MySQL

Запрос данных выполняется с помощью команды MySQL `SELECT`. Оператор `SELECT` имеет следующий формат:

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

Часть оператора с условиями является необязательной (рассмотрим ее позже). По сути, требуется знать имена столбцов и имя таблицы, из которой извлекаются данные.
Например, чтобы извлечь имена и фамилии всех сотрудников, выполните следующую команду.

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

Чтобы вывести всю таблицу, можно либо ввести имена всех столбцов, либо воспользоваться упрощенной формой оператора `SELECT`.

```
SELECT * from employee_data;
```

Символ * в этом выражении означает 'ВСЕ столбцы'. Поэтому этот оператор выводит все строки всех столбцов.

Выборка данных с помощью условий

Теперь более подробно рассмотрим формат оператора `SELECT`. Его полный формат имеет вид:

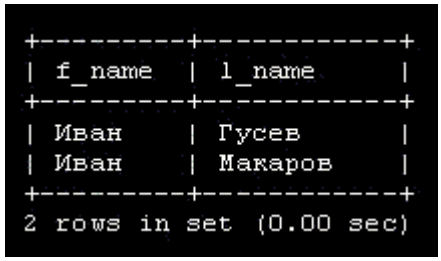
```
SELECT имена_столбцов from имя_таблицы [WHERE ...условия];
```

В операторе `SELECT` условия являются необязательными. Оператор `SELECT` без условий выводит все данные из указанных столбцов.

Операторы сравнения = и !=

```
SELECT f_name, l_name from employee_data where f_name = 'Иван';
```

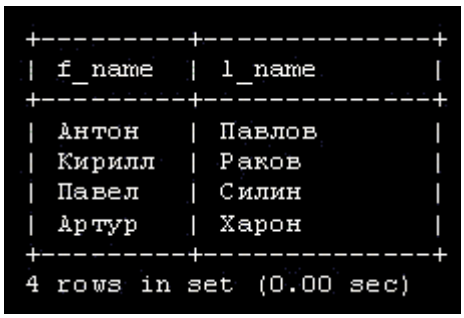
Результат запроса приведен на рисунке.
Этот оператор выводит имена и фамилии всех сотрудников, которые имеют имя Иван. Отметим, что слово Иван в условии заключено в одиночные кавычки. Можно использовать также двойные кавычки. Кавычки являются обязательными, так как MySQL будет порождать ошибку при их отсутствии. Кроме того сравнения MySQL не различают регистр символов, что означает, что с равным успехом можно использовать "Иван", "иван" и даже "ИвАн".



```
+---------+------------+
| f_name  | l_name     |
+---------+------------+
| Иван    | Гусев      |
| Иван    | Макаров    |
+---------+------------+
2 rows in set (0.00 sec)
```

```
SELECT f_name,l_name from employee_data where title="программист";
```

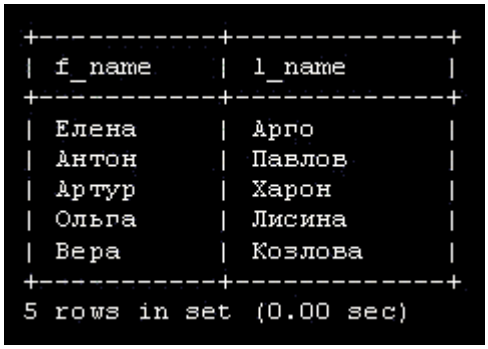
Результат запроса приведен на рисунке.
Выбирает имена и фамилии всех сотрудников, которые являются программистами.



```
+---------+--------------+
| f_name  | l_name       |
+---------+--------------+
| Антон   | Павлов       |
| Кирилл  | Раков        |
| Павел   | Силин        |
| Артур   | Харон        |
+---------+--------------+
4 rows in set (0.00 sec)
```

```
SELECT f_name, l_name from employee_data where age = 32;
```

Результат запроса приведен на рисунке.
Это список имен и фамилий всех сотрудников с возрастом 32 года. Вспомните, что тип столбца `age` был задан как `int`, поэтому кавычки вокруг 32 не требуются. Это - незначительное различие между текстовым и целочисленным типами столбцов.



```
+-----------+-------------+
| f_name    | l_name      |
+-----------+-------------+
| Елена     | Арго        |
| Антон     | Павлов      |
| Артур     | Харон       |
| Ольга     | Лисина      |
| Вера      | Козлова     |
+-----------+-------------+
5 rows in set (0.00 sec)
```

Оператор `!=` означает 'не равно' и является противоположным оператору равенства.

Операторы больше и меньше

Давайте получим имена и фамилии всех сотрудников, которые старше 32 лет.

```
SELECT f_name, l_name from employee_data where age > 32;
```

Результат запроса приведен на рисунке

```
+------------+-------------+
| f_name     | l_name      |
+------------+-------------+
| Федер      | Круглов     |
| Иван       | Макаров     |
| Эдуард     | Сахаров     |
| Кирилл     | Раков       |
| Павел      | Силин       |
| Римма      | Чащина      |
| Рита       | Белова      |
| Денис      | Горшков     |
| Елена      | Кароян      |
| Василий    | Курица      |
| Петр       | Ключник     |
+------------+-------------+
10 rows in set (0.00 sec)
```

Попробуем найти сотрудников, которые получают зарплату больше 120000.

```
SELECT f_name, l_name from employee_data where salary > 120000;
```

Результат запроса приведен на рисунке

```
+----------+----------+
| f_name   | l_name   |
+----------+----------+
| Михаил   | Петров   |
+----------+----------+
1 row in set (0.00 sec)
```

Теперь перечислим всех сотрудников, которые имеют стаж работы в компании менее 3 лет.

```
SELECT f_name, l_name from employee_data where yos < 3;
```

Результат запроса приведен на рисунке

```
+-----------+-------------+
| f_name    | l_name      |
+-----------+-------------+
| Рита      | Белова      |
| Денис     | Горшков     |
| Ольга     | Лисина      |
| Елена     | Кароян      |
| Василий   | Курица      |
| Вера      | Козлова     |
| Степан    | Аидов       |
| Петр      | Ключник     |
+-----------+-------------+
8 rows in set (0.00 sec)
```

Операторы <= и >=

Используемые в основном с целочисленными данными операторы меньше или равно (<=) и больше или равно (>=) обеспечивают дополнительные возможности.

```
select f_name, l_name, age, salary
from employee_data where age >= 32;
```

Результат запроса приведен на рисунке

```
+------------+-------------+-------+----------+
| f_name     | l_name      | age   | salary   |
+------------+-------------+-------+----------+
| Елена      | Арго        |  32   | 80000    |
| Федер      | Круглов     |  33   | 90000    |
| Иван       | Макаров     |  34   | 85000    |
| Эдуард     | Сахаров     |  35   | 100000   |
| Антон      | Павлов      |  32   | 90000    |
| Кирилл     | Раков       |  37   | 120000   |
| Павел      | Силин       |  38   | 120000   |
+------------+-------------+-------+----------+
7 rows in set (0.00 sec)
```

Выборка содержит имена, возраст и зарплаты сотрудников, которым больше 32 лет.

```
select f_name, l_name from employee_data where yos <= 2;
```

Результат запроса приведен на рис. 5.11.

```
+------------+------------+
| f_name     | l_name     |
+------------+------------+
| Федер      | Круглов    |
| Иван       | Макаров    |
| Эдуард     | Сахаров    |
| Антон      | Павлов     |
| Кирилл     | Раков      |
| Павел      | Силин      |
| Артур      | Харон      |
| Римма      | Чащина     |
| Рита       | Белова     |
| Денис      | Горшков    |
+------------+------------+
10 rows in set (0.00 sec)
```

Запрос выводит имена сотрудников, которые работают в компании меньше 3 лет.

Поиск текстовых данных по шаблону

В данной части мы рассмотрим *поиск текстовых данных по шаблону* с помощью предложения `where` и оператора `LIKE`.

Оператор сравнения на равенство (`=`) помогает выбрать одинаковые строки. Таким образом, чтобы перечислить имена сотрудников, которых зовут Иван, можно воспользоваться следующим оператором `SELECT`.

```
select f_name, l_name from employee_data where f_name = "Иван";
```

Результат запроса приведен на рисунке

```
+---------+-------------+
| f_name  | l_name      |
+---------+-------------+
| Иван    | Гусев       |
| Иван    | Макаров     |
+--------+--------------+
2 rows in set (0.00 sec)
```

Как быть, если надо вывести данные о сотрудниках, имя которых начинается с буквы В? Язык SQL позволяет выполнить поиск строковых данных по шаблону. Для этого в предложении `where` используется оператор `LIKE` следующим образом.

```
select f_name, l_name from employee_data where f_name LIKE "В%";
```

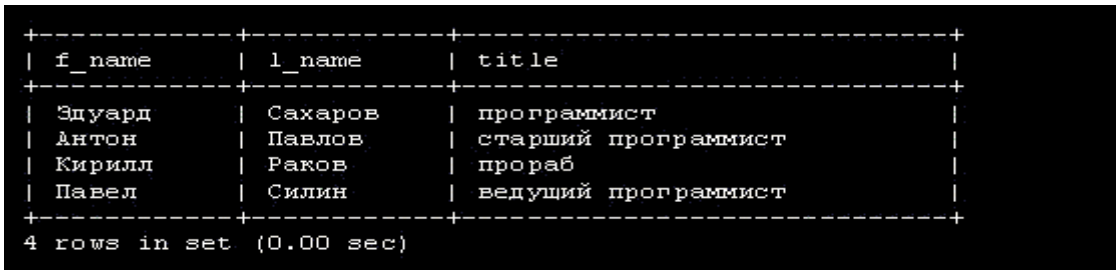
Результат запроса приведен на рисунке

```
+------------+------------+
| f_name     | l_name     |
+------------+------------+
| Василий    | Курица     |
| Вера       | Козлова    |
+------------+------------+
2 rows in set (0.00 sec)
```

Можно видеть, что здесь в условии вместо знака равенства используется `LIKE` и знак процента в шаблоне. Знак % действует как символ-заместитель. Он заменяет собой любую последовательность символов. Таким образом `"В%"` обозначает все строки, которые начинаются с буквы В. Аналогично `"%В"` выбирает строки, которые заканчиваются символом В, а `"%В%"` строки, которые содержат букву В.
Давайте выведем, например, всех сотрудников, которые имеют в названии должности строку "про".

```
select f_name, l_name, title from employee_data
where title like '%про%';
```

Результат запроса приведен на рисунке

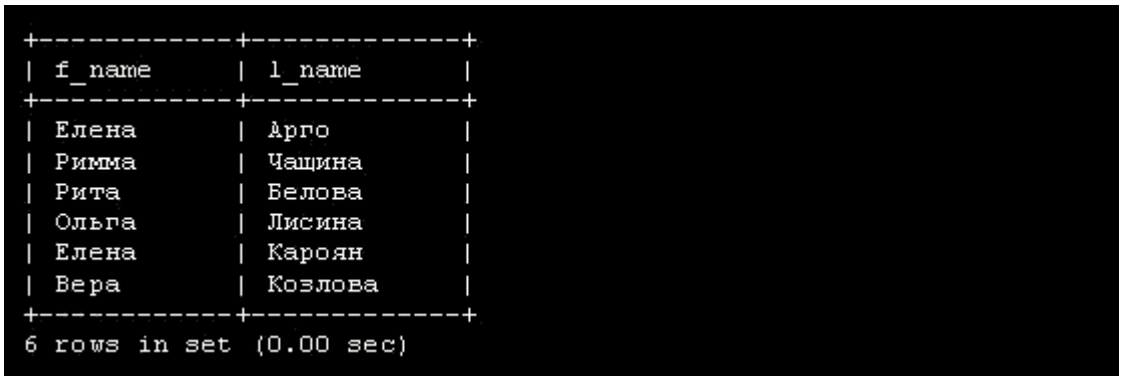


```
+------------+------------+------------------------------+
| f_name     | l_name     | title                        |
+------------+------------+------------------------------+
| Эдуард     | Сахаров    | программист                  |
| Антон      | Павлов     | старший программист          |
| Кирилл     | Раков      | прораб                       |
| Павел      | Силин      | ведущий программист          |
+------------+------------+------------------------------+
4 rows in set (0.00 sec)
```

Перечислим всех сотрудников, имена которых заканчиваются буквой 'а'. Это очень просто сделать.

```
mysql> select f_name, l_name from employee_data
where f_name like '%а';
```

Результат запроса приведен на рисунке



```
+------------+-------------+
| f_name     | l_name      |
+------------+-------------+
| Елена      | Арго        |
| Римма      | Чащина      |
| Рита       | Белова      |
| Ольга      | Лисина      |
| Елена      | Кароян      |
| Вера       | Козлова     |
+------------+-------------+
6 rows in set (0.00 sec)
```

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;
```

Результат запроса приведен на рисунке

```
+----------------------------------------+----------------+
| title                                  | AVG(salary)    |
+----------------------------------------+----------------+
| директор                               | 200000         |
| менеджер по работе с заказчиком        |  70000         |
| бухгалтер                              | 120000         |
| продавец                               |  77000         |
| программист мультимедиа                |  83000         |
| программист                            |  75000         |
| старший продавец                       | 120000         |
| старший программист                    | 115000         |
| старший разработчик Web                | 110000         |
| системный администратор                |  95000         |
| разработчик Web                        |  87500         |
+----------------------------------------+----------------+
11 rows in set (0.00 sec)
```

Предположим теперь, что требуется вывести только те подразделения, где средняя зарплата более 100000. Это можно сделать с помощью предложения `HAVING`.

```
select title, AVG(salary)
from employee_data
GROUP BY title
HAVING AVG(salary) > 100000;
```

Результат запроса приведен на рисунке

```
+----------------------------------------+----------------+
| title                                  | AVG(salary)    |
+----------------------------------------+----------------+
| директор                               | 200000         |
| бухгалтер                              | 120000         |
| старший продавец                       | 120000         |
| старший программист                    | 115000         |
| старший разработчик Web                | 110000         |
+----------------------------------------+----------------+
5 rows in set (0.00 sec)
```

Удаление записей из таблицы

Для *удаления записей из таблицы* можно использовать оператор `DELETE`.

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

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

Примечание: Если никакие условия не будут заданы, то удаляются все данные в таблице.

Предположим, один из специалистов по мультимедиа 'Василий Пупкин' уволился из компании. Надо удалить его запись.

```
DELETE from employee_data
```

```
WHERE l_name = 'Пупкин';
```

Результат запроса приведен на рисунке

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

Логические операторы

Рассмотрим, как выбрать данные на основе условий SQL, представленных с помощью *булевых (логических) операторов*.

1. AND
2. OR
3. NOT

Использовать их очень просто. Ниже показан оператор `SELECT`, который выводит имена сотрудников, которые получают более 70000, но меньше 90000.

```
SELECT f_name, l_name from employee_data
where salary > 70000 AND salary < 90000;
```

На рисунке приведен результат запроса.

```
+-----------+-------------+
| f_name    | l_name      |
+-----------+-------------+
| Иван      | Макаров     |
| Эдуард    | Сахаров     |
| Кирилл    | Раков       |
| Павел     | Силин       |
| Римма     | Чащина      |
| Рита      | Белова      |
| Денис     | Горшков     |
+-----------+-------------+
7 rows in set (0.00 sec)
```

```
+------------+
| l_name     |
+------------+
| Круглов    |
| Лисина     |
| Кароян     |
| Курица     |
| Козлова    |
| Ключник    |
+------------+
6 rows in set (0.00 sec)
```

Давайте выведем список сотрудников, фамилии которых начинаются с буквы К или Л.

```
SELECT l_name from employee_data where
l_name like 'К%' OR l_name like 'Л%';
```

На рисунке приведен результат запроса.

Вот более сложный пример ... список имен и возраста сотрудников, фамилии которых начинаются с К или Л, и которые младше 30 лет.

```
SELECT f_name, l_name , age from employee_data
where (l_name like 'К%' OR l_name like 'Л%') AND age < 30;
```

```
+-----------+------------+------+
| f_name    | l_name     | age  |
+-----------+------------+------+
| Иван      | Макаров    |  28  |
| Эдуард    | Сахаров    |  26  |
| Кирилл    | Раков      |  25  |
| Павел     | Силин      |  27  |
+-----------+------------+------+
4 rows in set (0.00 sec)
```

Обратите внимание на использование скобок в представленном выше операторе. Скобки предназначены для выделения различных логических условий и удаления двусмысленностей.

Оператор `NOT` поможет при поиске всех сотрудников, которые не являются программистами. (Программисты включают старших программистов, программистов мультимедиа и программистов).

```
SELECT f_name, l_name, title from employee_data
where title NOT LIKE "%программист%";
```

Операторы IN и BETWEEN

Чтобы найти сотрудников, которые являются разработчиками Web или системными администраторами, можно использовать оператор `SELECT` следующего вида:

```
SELECT f_name, l_name, title from
    -> employee_data where
    -> title = 'разработчик Web' OR
    -> title = 'системный адм.';
```

На рисунке приведен результат запроса.

```
+-----------+------------+-----------------------+
| f_name    | l_name     | title                 |
+-----------+------------+-----------------------+
| Иван      | Макаров    |  разработчик Web      |
| Эдуард    | Сахаров    |  разработчик Web      |
| Кирилл    | Раков      |  системный адм.       |
| Павел     | Силин      |  системный адм.       |
+-----------+------------+-----------------------+
4 rows in set (0.01 sec)
```

В SQL имеется более простой способ сделать это с помощью *оператора* `IN` (в множестве). Его использование не представляет никаких трудностей.

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

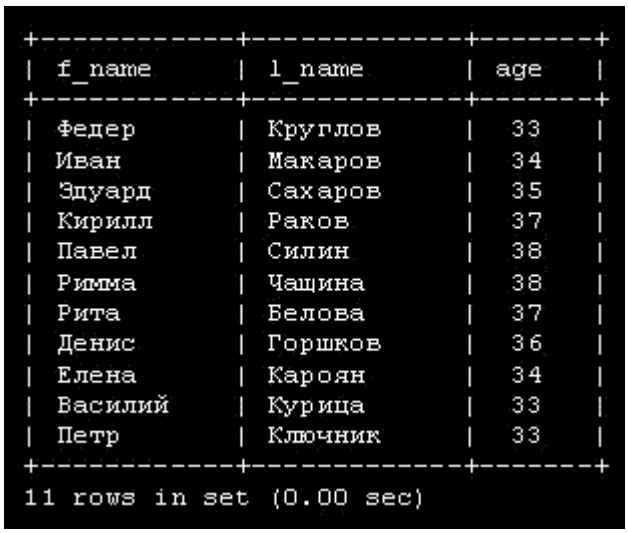
Результат будет аналогичен прошлому примеру.
Использование `NOT` перед `IN` позволяет вывести данные, которые не входят в множество, определяемое условием `IN`. Следующий оператор выводит список сотрудников, которые не занимают должность программиста или системного администратора.

```
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;
```

На рисунке приведен результат запроса.



f_name	l_name	age
Федер	Круглов	33
Иван	Макаров	34
Эдуард	Сахаров	35
Кирилл	Раков	37
Павел	Силин	38
Римма	Чащина	38
Рита	Белова	37
Денис	Горшков	36
Елена	Кароян	34
Василий	Курица	33
Петр	Ключник	33

11 rows in set (0.00 sec)

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;
```

Упорядочивание данных

Рассмотрим вопрос о том, как можно изменить *порядок вывода данных*, извлеченных из таблиц MySQL, используя предложение ORDER BY оператора SELECT.

Извлекаемые до сих пор данные всегда выводились в том порядке, в котором они были сохранены в таблице. В действительности SQL позволяет сортировать извлеченные данные с помощью предложения ORDER BY. Это предложение требует имя столбца, на основе которого будут сортироваться данные. Давайте посмотрим, как можно вывести имена сотрудников с упорядоченными по алфавиту фамилиями сотрудников (в возрастающем порядке).

```
SELECT l_name, f_name from
employee_data ORDER BY l_name;
```

А вот так сотрудников можно отсортировать по возрасту.

```
SELECT f_name, l_name, age
from employee_data
ORDER BY age;
```

Предложение ORDER BY может сортировать в возрастающем порядке (ASCENDING или ASC) или в убывающем порядке (DESCENDING или DESC) в зависимости от указанного аргумента.

Чтобы вывести список сотрудников в убывающем порядке, можно использовать следующий оператор.

```
SELECT f_name from employee_data
ORDER by f_name DESC;
```

Примечание: Возрастающий порядок (ASC) используется по умолчанию.

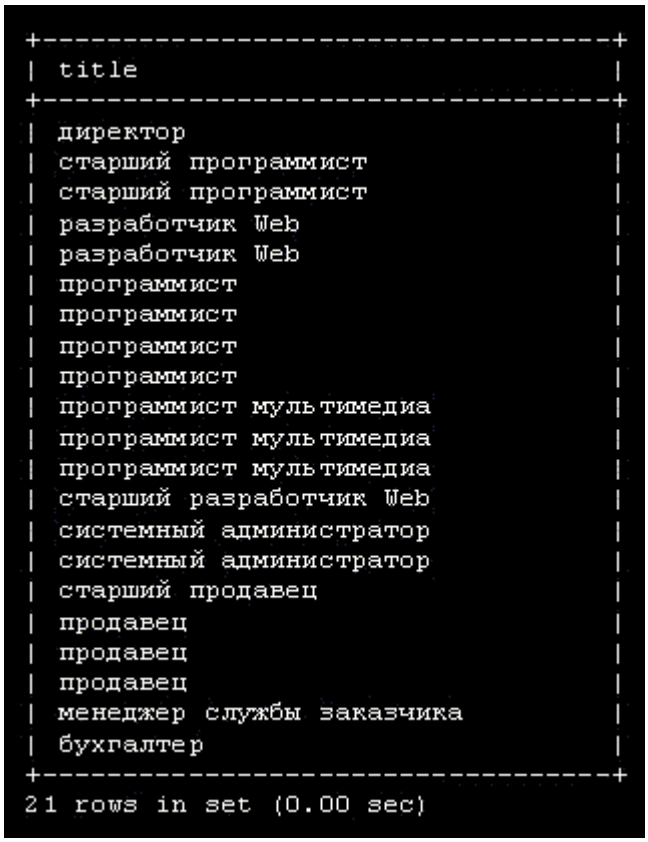
Ключевое слово DISTINCT

Рассмотрим теперь, как выбрать и вывести записи таблиц MySQL с помощью *ключевого слова* `DISTINCT` (РАЗЛИЧНЫЙ), использование которого исключает появление повторяющихся данных.

Чтобы вывести все должности базы данных компании, можно выполнить следующий оператор:

```
select title from employee_data;
```

На рисунке приведен результат запроса.

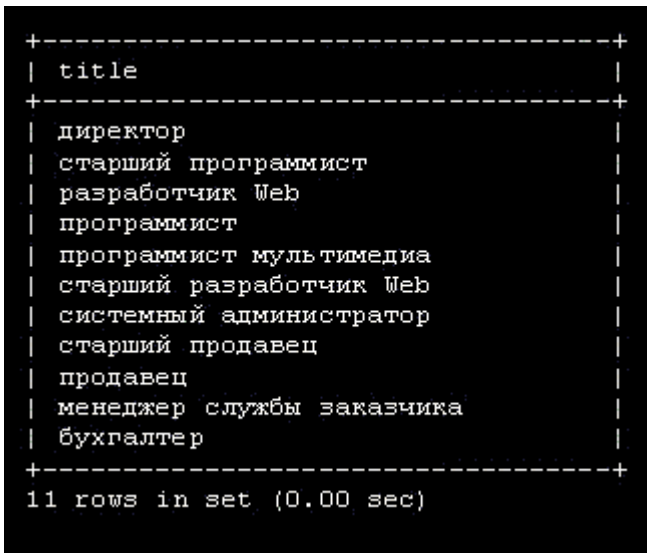


```
+-----------------------------------+
| title                             |
+-----------------------------------+
| директор                          |
| старший программист               |
| старший программист               |
| разработчик Web                   |
| разработчик Web                   |
| 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.



Из этого можно видеть, что в компании имеется 11 уникальных должностей.
Уникальные записи можно также отсортировать с помощью `ORDER BY`.

```
select DISTINCT age from employee_data
ORDER BY age;
```

Изменение записей

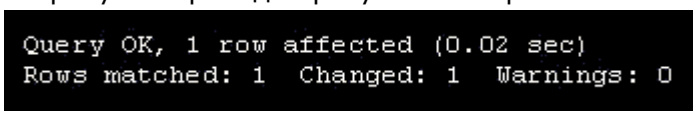
Команда `UPDATE` выполняет *изменение данных* в таблицах. Она имеет очень простой формат.

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

Как и все другие команды SQL можно вводить ее на одной строке или на нескольких строках.
Рассмотрим несколько примеров.
Предположим, директор увеличил свою зарплату на 20000 и надбавки на 5000. Его предыдущая зарплата была 200000, а надбавки были 50000.

```
UPDATE employee_data SET
salary=220000, perks=55000
WHERE title='директор';
```

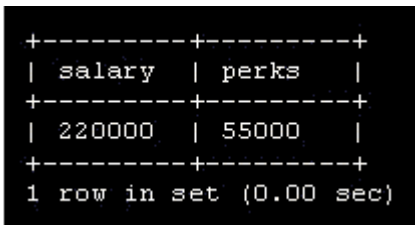
На рисунке приведен результат запроса.



Можно проверить эту операцию, выводя данные из таблицы.

```
select salary, perks from
employee_data WHERE
title = 'директор';
```

На рисунке приведен результат запроса.



В действительности предыдущую зарплату знать не требуется. Можно воспользоваться арифметическими операторами. Следующий оператор сделает то же самое, при этом исходные данные знать заранее не требуется.

```
UPDATE employee_data SET
salary = salary + 20000,
perks = perks + 5000
WHERE title='директор';
```

В качестве другого примера можно попробовать изменить название должности "разработчик Web" на "программист Web".

```
mysql> update employee_data SET
    -> title = 'программист Web'
    -> WHERE title = 'разработчик Web';
```

Важно также перед выполнением изменений внимательно изучить часть оператора с условием, так как легко можно изменить не те данные. Оператор `UPDATE` без условий изменит все данные столбца во всех строках. Надо быть очень осторожным при внесении изменений.

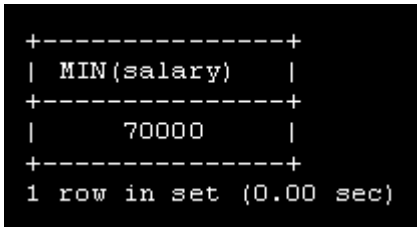
Команды обработки данных

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

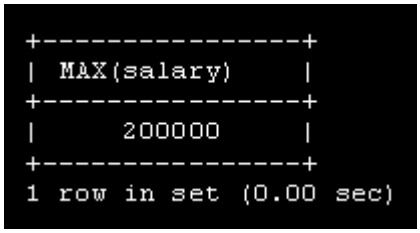
Минимальное значение

`select MIN(salary) from employee_data;`
На рисунке приведен результат запроса.



Максимальное значение

`select MAX(salary) from employee_data;`
На рисунке приведен результат запроса.



Поиск среднего значения и суммы

Суммирование значений столбца с помощью функции SUM

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

Давайте посмотрим, сколько компания тратит на зарплату своих сотрудников.

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

```
+---------------+
| SUM(salary)   |
+---------------+
|     1997000   |
+---------------+
1 row in set (0.00 sec)
```

Аналогично можно вывести общую сумму надбавок, выдаваемых сотрудникам.

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

```
+---------------+
| SUM(perks)    |
+---------------+
|     390000    |
+---------------+
1 row in set (0.00 sec)
```

Можно найти также общую сумму зарплаты и надбавок.

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

```
+------------------------------+
| sum(salary)+ sum(perks)      |
+------------------------------+
|           2387000            |
+------------------------------+
1 row in set (0.01 sec)
```

Вычисление среднего значения

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

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

На рисунке приведен результат запроса.

```
+----------+
| avg(age) |
+----------+
|  31.6190 |
+----------+
1 row in set (0.00 sec)
```

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

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

На рисунке приведен результат запроса.

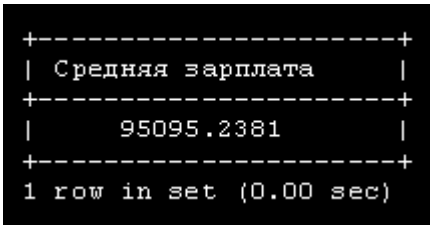
```
+---------------+
| avg(salary)   |
+---------------+
|  95095.2381   |
+---------------+
1 row in set (0.00 sec)
```

Именование столбцов

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;
```

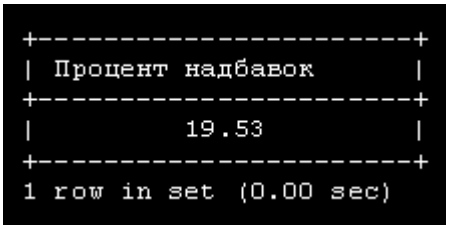
На рисунке приведен результат запроса.



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 рисунке приведен результат запроса.

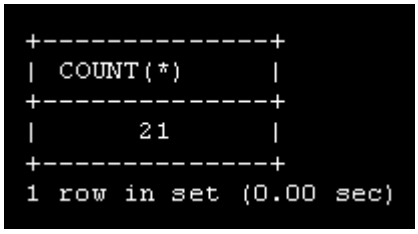


Подсчет числа записей

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;
```

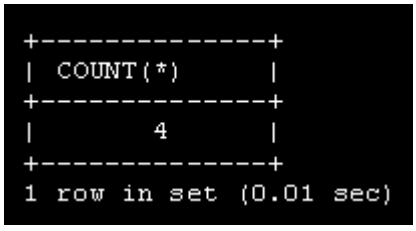
На рисунке приведен результат запроса.



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 рисунке приведен результат запроса.

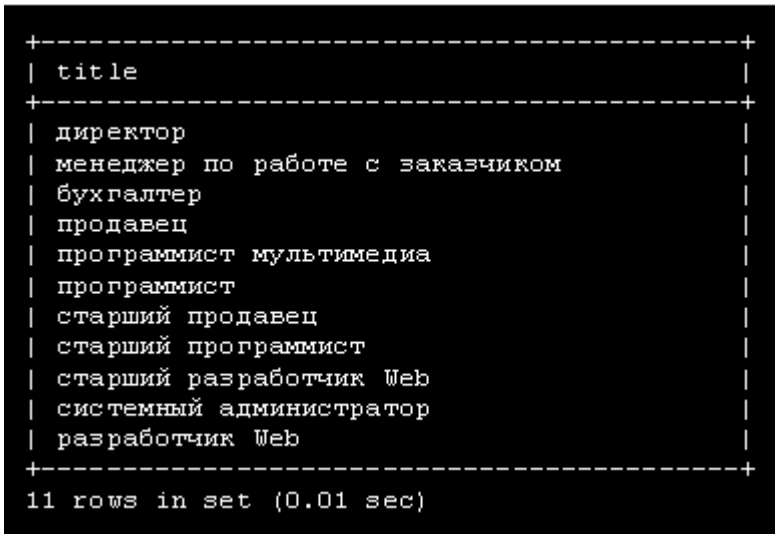


Группировка данных

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 рисунке приведен результат запроса.

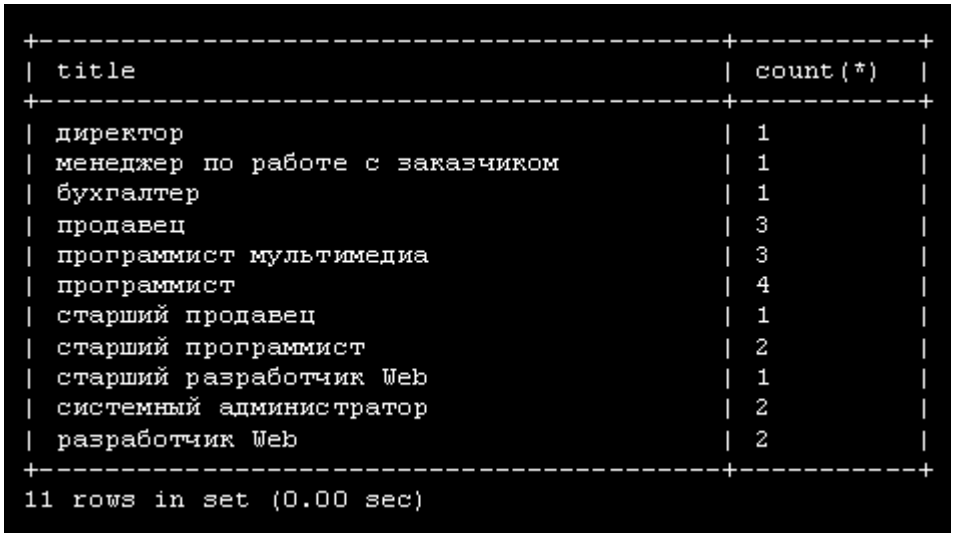


```
+----------------------------------------+
| title                                  |
+----------------------------------------+
| директор                               |
| менеджер по работе с заказчиком        |
| бухгалтер                              |
| продавец                            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

Вот как можно подсчитать число сотрудников имеющих определенную должность.

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

На рисунке 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)
```

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

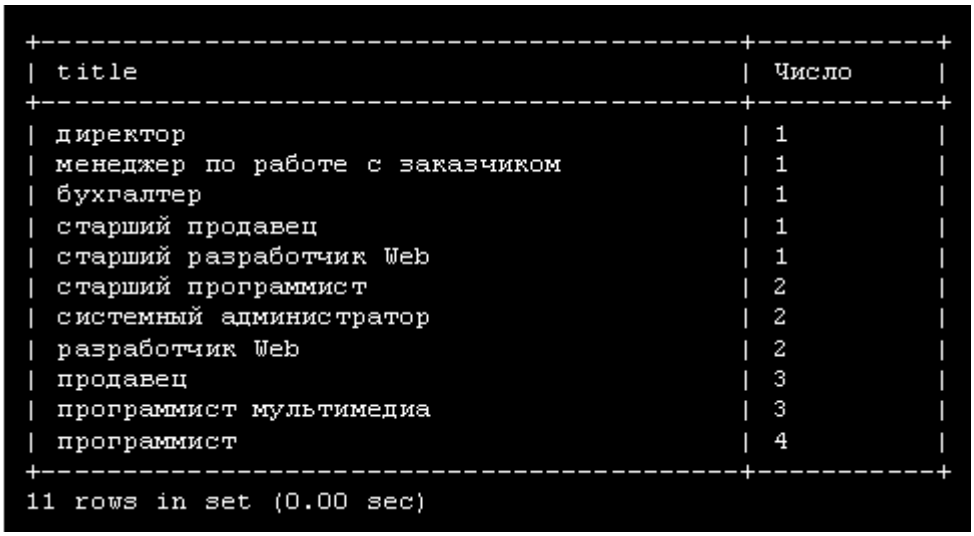
Сортировка данных

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 рисунке приведен результат запроса.



title	Число
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)

Задания

1. Напишите оператор SQL для создания новой базы данных с именем `addressbook`
2. Напишите оператор для записи следующих данных в таблицу `employee_data`

   ```
   Имя: Рудольф
   Фамилия: Курочкин
   Должность: Программист
   Возраст: 34
   Стаж работы в компании: 2
   Зарплата: 95000
   Надбавки: 17000
   email: rudolf@yandex.ru
   ```
3. Напишите оператор для вывода данных из столбцов `salary`, `perks` и `yos` таблицы `employee_data`.
4. Как извлечь данные столбцов `salary`, `l_name` из таблицы `employee_data`?
5. Напишите оператор `SELECT` для извлечения идентификационного номера сотрудников, которые старше 30 лет.
6. Напишите оператор `SELECT` для извлечения имен и фамилий всех Web-разработчиков.
7. Как вывести зарплаты и надбавки сотрудников, которые получают в качестве надбавок более 16000?
8. Перечислите имена всех сотрудников (фамилия, а затем имя), которые занимают должность бухгалтера.
9. Перечислить всех сотрудников, фамилии которых начинаются с буквы Р.
10. Вывести имена всех сотрудников в отделе продаж.
11. Перечислить фамилии и должности всех программистов
12. Вывести подразделения и средний возраст, где средний возраст больше 30.
13. Вывести имена и фамилии всех сотрудников, которые получают зарплату не более 90000 и не являются программистами, старшими программистами или программистами мультимедиа.
14. Вывести все идентификационные номера и имена сотрудников в возрасте от 32 до 40 лет.
15. Выберите имена всех сотрудников в возрасте 32 лет, которые не являются программистами.
16. Найдите всех сотрудников, которые занимают должность "старший программист" и "программист мультимедиа".
17. Выведите список имен сотрудников, зарплата которых составляет от 70000 до 90000.
18. Вывести список сотрудников в порядке, определяемом зарплатой, которую они получают.
19. Выведите список сотрудников в убывающем порядке их стажа работы в компании.
20. Вывести список сотрудников (фамилию и имя), которые занимают должность "программист" или "разработчик Web" и отсортировать их фамилии по алфавиту.
21. Сколько различных имен имеется в базе данных?
22. Измените фамилию Чащина на Петрова. Внесите соответствующие изменения в базу данных.
23. Название должности "программист мультимедиа" необходимо изменить на "специалист по мультимедиа".
24. Увеличьте зарплату всем сотрудниками (кроме директора) на 10000.
25. Найдите минимальные надбавки.
26. Найдите максимальную зарплату среди всех "программистов".
27. Найдите возраст самого старого "продавца".

28. Найдите имя и фамилию самого старого сотрудника.
29. Вывести сумму всех возрастов сотрудников, работающих в компании.
30. Как вычислить общее количество лет стажа работы сотрудников в компании?
31. Вычислите сумму зарплат и средний возраст сотрудников, которые занимают должность "программист".
32. Подсчитайте число сотрудников, которые проработали в компании более трех лет.
33. Найдите средний возраст сотрудников в различных подразделениях (должностях).
34. Измените предыдущий оператор так, чтобы данные выводились в убывающем порядке среднего возраста.