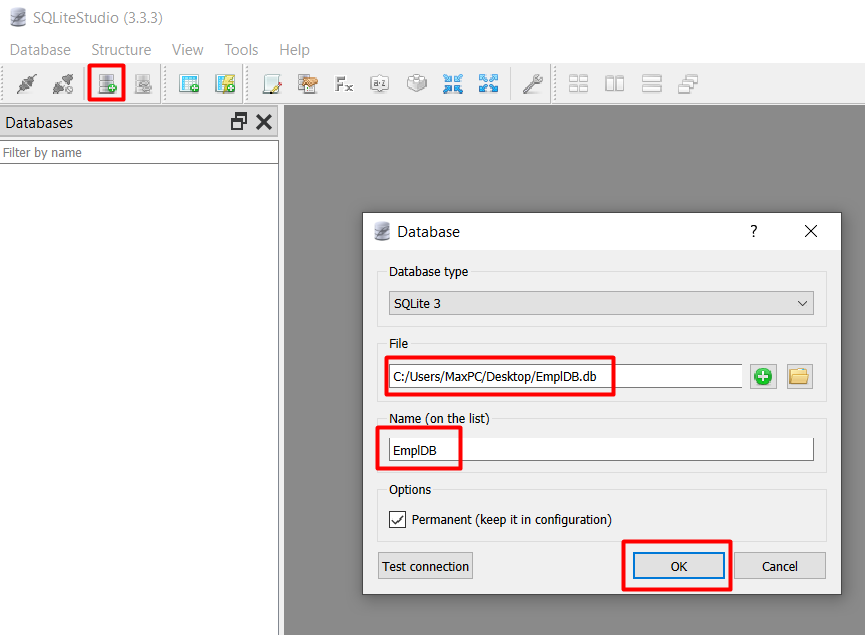
**Лаб. работа №1 Создание базы данных SQLite**

Для создания новой базы данных мы можем либо воспользоваться командной строкой, идущей в комплекте с архивом SQLite, либо использовать инструмент SQLite Studio (<https://sqlitestudio.pl/>), в котором есть редактор запросов.

Откройте SQLite Studio -> Иконка «Создать базу данных» -> Выберите расположение базы данных на диске -> Введите имя базы данных -> Нажмите «ОК».

Для создания таблиц можно либо использовать конструктор таблиц, либо создавать таблицы, написав запросы к базе данных. Воспользуемся вторым вариантом — редактором запросов.



Откройте редактор запросов SQL (значок с пустым листом и карандашом) и введите следующий код:

*CREATE TABLE Employees (*

*id integer PRIMARY KEY AUTOINCREMENT,*

*name varchar(50),*

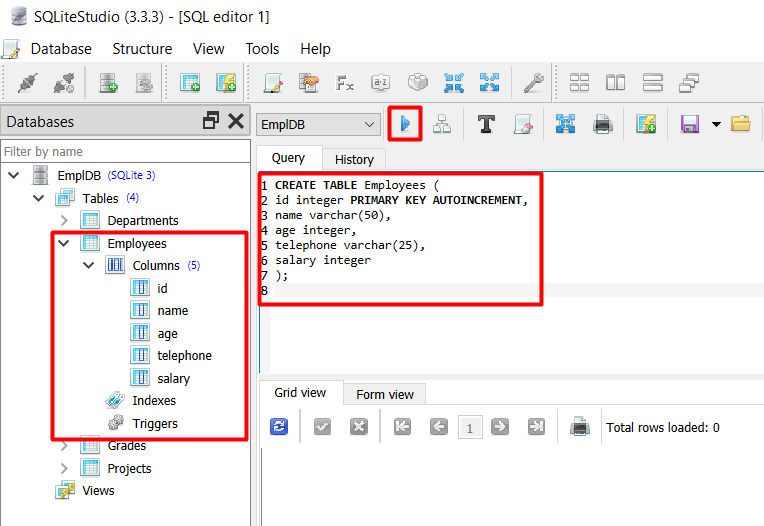
*age integer,*

*telephone varchar(25),*

*salary integer*

*);*

Нажмите кнопку с синим треугольником. Запрос будет отправлен в базу данных. Результатом запроса будет новая таблица в базе данных (см. рис. ниже).



Давайте создадим еще несколько таблиц, которые будут связаны друг с другом с помощью внешних ключей. В редакторе запросов введите следующий код и нажмите на синий треугольник (выполнение запроса):

*CREATE TABLE Departments (*

*id integer PRIMARY KEY AUTOINCREMENT,*

*name varchar(100)*

*);*

Затем добавьте таблицу Projects:

*CREATE TABLE Projects (*

*id integer PRIMARY KEY AUTOINCREMENT,*

*start\_date varchar(15),*

*end\_date varchar(15),*

*employees\_amount integer,*

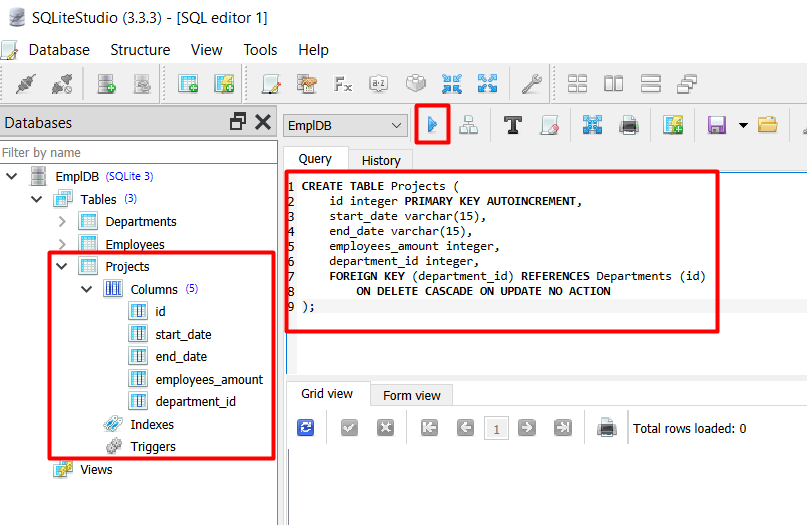
*department\_id integer,*

*FOREIGN KEY (department\_id) REFERENCES Departments (id)*

*ON DELETE CASCADE ON UPDATE NO ACTION*

*);*

В таблице выше есть внешний ключ «department\_id», который свяжет таблицы «Projects» и «Departments».



Мы также добавим четвертую таблицу, которая будет иметь два внешних ключа:

*CREATE TABLE Grades (*

*project\_id integer,*

*employee\_id integer,*

*grade integer,*

*FOREIGN KEY (project\_id) REFERENCES Projects (id)*

*ON DELETE CASCADE ON UPDATE NO ACTION,*

*FOREIGN KEY (employee\_id) REFERENCES Employees (id)*

*ON DELETE CASCADE ON UPDATE NO ACTION*

*);*

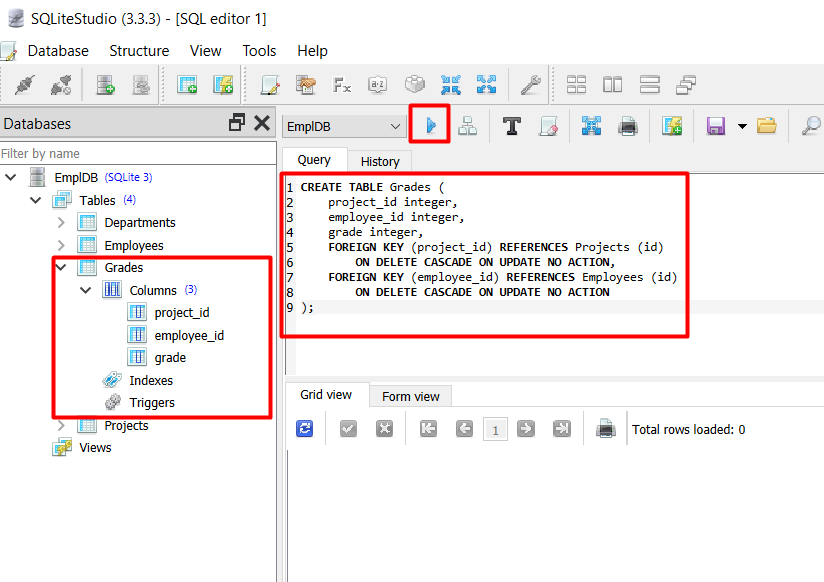
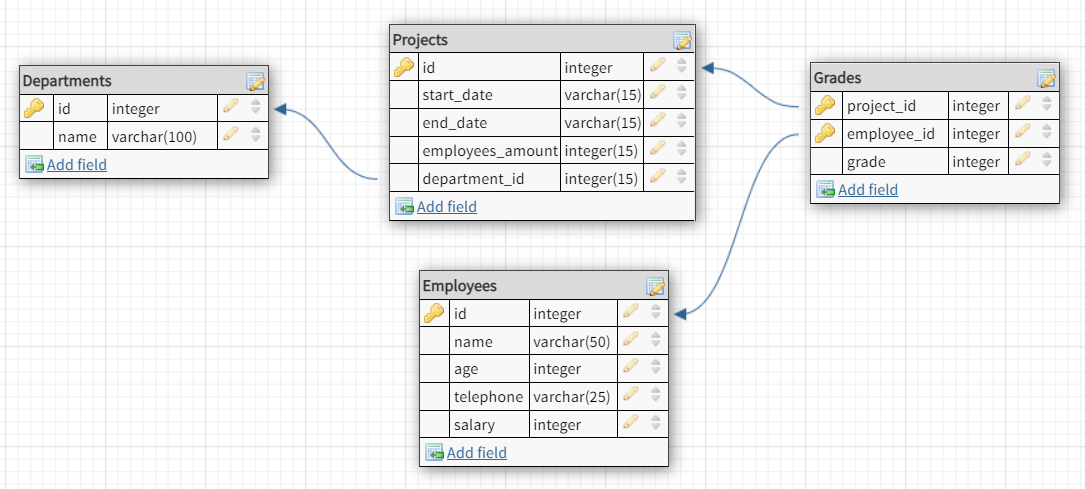


Схема данных в БД после добавления таблиц будет выглядеть так:



На схеме показано, как таблицы связаны между собой.

Далее рассмотрим основные команды SQL на примере работы с таблицей «Employees».

**CREATE statement**

Команда CREATE используется для создания новой таблицы в базе данных. Мы использовали эту команду выше при создании таблиц в базе данных.

**SELECT statement**

Основной целью работы с системами управления базами данных является не только хранение данных, но и возможность организовать доступ к этим данным и манипулировать ими.

Допустим, после создания таблицы в реляционной базе данных и добавления данных в таблицу мы хотим получить данные. Чтобы получить данные из таблицы, мы должны использовать оператор SELECT.

SELECT — это оператор языка манипулирования данными (DML). Операторы DML используются для чтения и изменения данных. Оператор SELECT создает запрос к базе данных и берет оттуда некоторые данные. Ответ, который мы получаем после выполнения этого запроса, называется «набором результатов» или «таблицей результатов».

В простейшей форме оператор SELECT выглядит так:

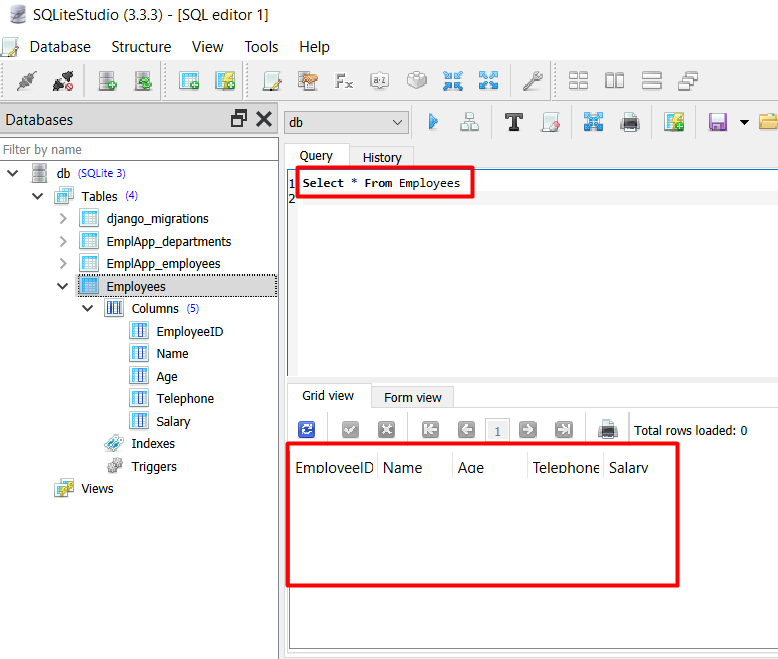
*SELECT \* FROM table\_name*

Для наглядного примера воспользуемся таблицей сотрудников компании с названием «Employees» (см. ниже).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **id** | **name** | **age** | **telephone** | **salary** |
| 1 | Peter | 23 | 325-986 | 4000 |
| 2 | John | 25 | 586-895 | 5000 |
| 3 | Will | 32 | 258-951 | 10000 |
| 4 | Denis | 20 | 753-854 | 5000 |
| 5 | Tony | 25 | 843-861 | 10000 |
| 6 | Bob | 29 | 266-826 | 4000 |
| 7 | Max | 31 | 377-657 | 7000 |

Используя оператор SELECT, выберем все данные из таблицы Employees.

*SELECT \* FROM Employees*



На этом этапе мы получим результирующую таблицу, которая будет содержать только названия столбцов, так как мы не заполняли таблицу реальными данными. Добавим в таблицу реальные данные. Для этого выполним следующий запрос с помощью оператора INSERT (мы поговорим о нем чуть позже):

*INSERT INTO Employees (name, age, telephone, salary)*

*VALUES*

*('Peter','23','325-986','4000'),*

*('John','25','586-895','5000'),*

*('Will','32','258-951','10000'),*

*('Denis','20','753-854','5000'),*

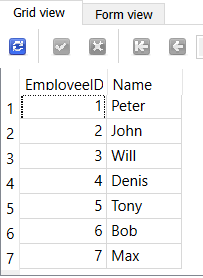
*('Tony','25','842-861','10000'),*

*('Bob','29','266-826','4000'),*

*('Max','31','377-657','7000')*

Однако, нам не всегда нужно брать все данные из таблицы. Поэтому мы можем написать запрос на получение только той части данных, которая нам нужна. Например, мы хотим получить только два столбца из таблицы Employees: id и name. Запрос будет выглядеть так:

*SELECT id, name FROM Employees*

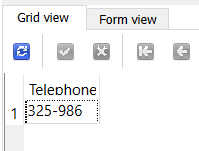


**Необходимо учитывать,** что порядок отображения столбцов такой же, как указанный в операторе SELECT.

А что если мы хотим узнать номер телефона сотрудника, чей ID равен 1?

В этом случае мы можем ограничить набор результатов, используя специальное условие WHERE. Оператор SELECT требует использования условия (предиката), по которому будет производиться выбор. При выполнении запроса к базе данных для предиката будет вычислено значение: true, false или unknown. Поэтому запрос будет выглядеть так:

*SELECT telephone FROM Employees WHERE id = 1*



В приведенном выше примере используется оператор сравнения «=». Также возможно использование других операторов, поддерживаемых системой управления реляционными базами данных:

|  |  |
| --- | --- |
| Равно | "=" |
| Больше, чем | > |
| Больше или равно | >= |
| Меньше | < |
| Меньше или равно | <= |
| Не равный | "=" |

**COUNT, DISTINCT, LIMIT**

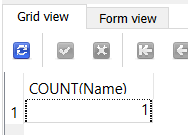
**COUNT**

В запросе с оператором SELECT вы также можете использовать дополнительные операторы для указания запроса.

COUNT — встроенная функция, позволяющая вернуть количество строк, удовлетворяющих определенному условию, например:

*SELECT COUNT (\*) FROM TABLE\_NAME*

*SELECT COUNT (name) FROM Employees WHERE name = 'Max'*



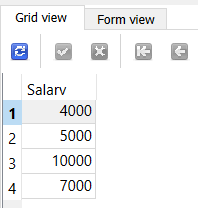
**DISTINCT**

DISTINCT — это функция, которая позволяет удалять повторяющиеся значения из результирующей таблицы. Например, запрос на получение уникальных значений в столбце будет выглядеть так:

*SELECT DISTINCT COLUMN\_NAME FROM TABLE\_NAME WHERE condition*

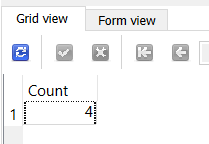
*SELECT DISTINCT salary FROM Employees*

Результатом запроса выше будет следующая таблица:



Например, чтобы посчитать все уникальные значения зарплаты из таблицы «Employees» (будет 4 значения: 4000, 5000, 7000, 10000), нужно написать следующий запрос:

*SELECT COUNT (DISTINCT salary) AS Count FROM Employees*



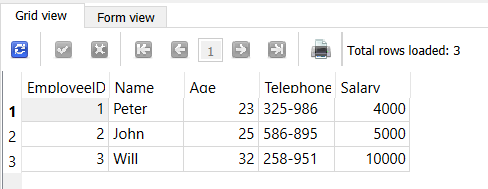
Оператор AS в запросе задает новое имя для поля при выборке из БД (в этом случае поле в результирующей таблице будет называться «Count»).

**LIMIT**

Функция LIMIT используется для ограничения количества строк, извлекаемых из базы данных.

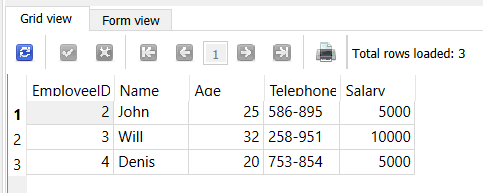
Например, чтобы получить первые три строки таблицы, запрос будет выглядеть так:

*SELECT \* FROM Employees WHERE id >0 LIMIT 3*



Если вам нужно выбрать три записи из таблицы, начиная со второй строки (первая строка имеет индекс 0, вторая строка имеет индекс 1):

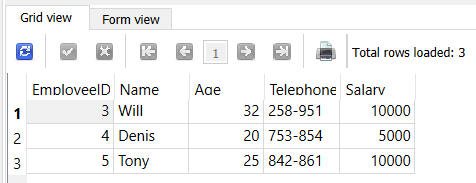
*SELECT \* FROM Employees WHERE id >0 LIMIT 1, 3*



**BETWEEN**

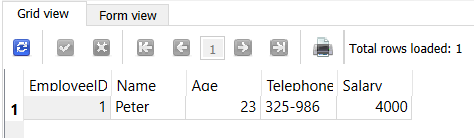
Функция BETWEEN позволяет выбирать записи из таблицы в заданном интервале, например:

*SELECT \* FROM Employees WHERE id between 2 and 4*



Вы также можете комбинировать различные условия. Пример запроса с условием:

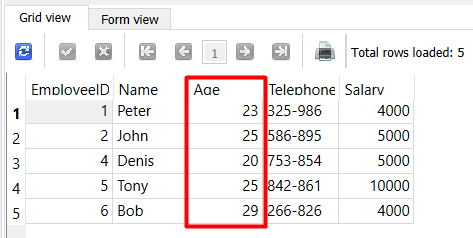
*SELECT \* FROM Employees WHERE id between 1 and 4 and salary=4000*



**LIKE**

Функция LIKE задает поиск определенного шаблона. Например, мы можем выбрать все записи из таблицы, где возраст начинается с цифры 2:

*SELECT \* FROM Employees WHERE age LIKE '2\_'*

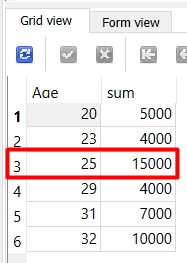


**GROUP BY**

Функция GROUP BY позволяет группировать результаты при выборке из базы данных. Допустим, мы хотим сгруппировать работников по возрасту, а затем применить функцию суммы, которая суммирует зарплаты в каждой группе. В результате для каждой из групп будет рассчитана общая заработная плата внутри этой группы. Запрос будет выглядеть так:

*SELECT age, SUM(salary) as sum FROM Employees GROUP BY age*

Результирующая таблица будет выглядеть следующим образом:



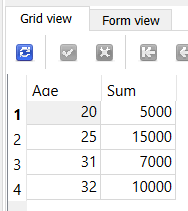
**HAVING**

Функция ИМЕЕТ позволяет фильтровать результаты группировки, выполненной с помощью функции GROUP BY. В целом схема запроса выглядит так:

*GROUP BY field HAVING condition*

Используем предыдущий запрос с функцией GROUP BY и добавляем к нему функцию HAVING, оставляя только те строки, в которых суммарная зарплата больше или равна 5000. Запрос будет выглядеть так:

*SELECT age, SUM(salary) as Sum FROM Employees GROUP BY age HAVING sum >=5000*



**INSERT Statement**

После того, как таблица в базе данных создана, в нее необходимо добавить данные. Чтобы вставить новые данные, мы используем оператор INSERT. Этот оператор добавляет новую строку данных в таблицу. Оператор INSERT — это оператор языка манипулирования данными (DML). Операторы DML используются для чтения и изменения данных.

Синтаксис для вставки данных может иметь несколько вариантов. Мы можем вставить одну или несколько записей сразу в таблицу данных. В целом синтаксис выглядит так:

*INSERT INTO Table\_Name (field 1, field 2, ...) VALUES (value 1, value 2, ...)*

Чтобы вставить сразу несколько записей, вы можете использовать запрос, как показано ниже:

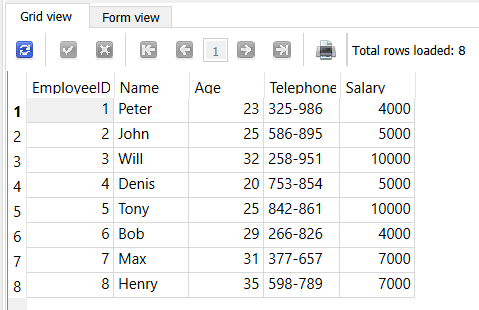
*INSERT INTO Table\_Name(field 1, field 2, ...)*

*VALUES (value 1, value 2, ...) , (value 1, value 2, ...)*

В примерах выше мы создали таблицу с данными о сотрудниках предприятия. Добавим в эту таблицу еще одну строку данных. Для этого необходимо использовать следующий запрос:

*INSERT INTO Employees (name, age, telephone, salary) VALUES ('Henry','35','598-789','7000')*

Если мы получим данные из таблицы с помощью оператора SELECT, результирующая таблица будет выглядеть так:

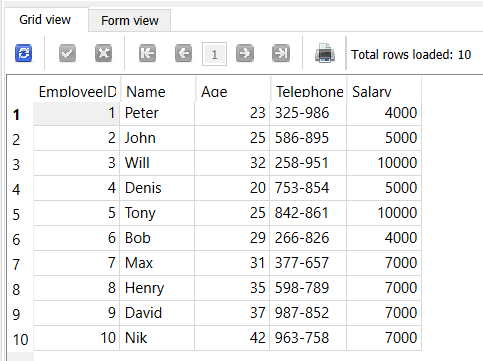


Добавим в таблицу сразу несколько строк. Для этого мы используем запрос со следующим синтаксисом:

*INSERT INTO Employees (name, age, telephone, salary)*

*VALUES ('David','37','987-852','7000'), ('Nik','42','963-758','7000')*

После добавления и запроса всех данных из таблицы мы получим следующую результирующую таблицу:



**UPDATE Statement**

Если есть необходимость обновить данные в любой строке таблицы, мы используем оператор UPDATE. Оператор UPDATE также является оператором языка манипулирования данными (DML), который позволяет нам манипулировать данными.

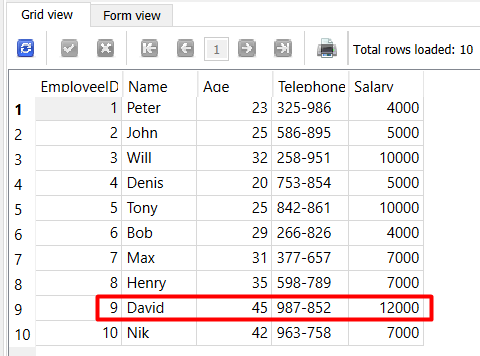
В общем случае запрос UPDATE будет выглядеть так:

*UPDATE [TableName] SET [ColumnName] = [Value] ] WHERE [Condition]*

В приведенном выше запросе TableName определяет имя таблицы, ColumnName указывает, какой столбец следует изменить в соответствии с условием WHERE.

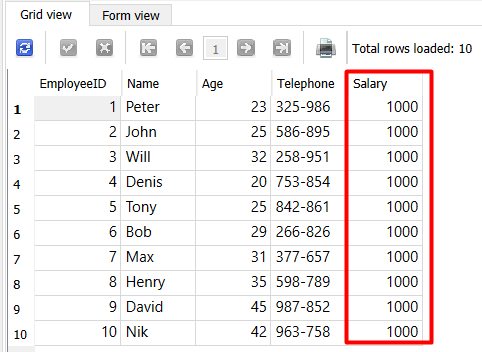
Допустим, после создания таблицы и добавления в нее данных мы хотим изменить данные одного из работников. Для этого мы используем запрос с оператором UPDATE следующим образом:

*UPDATE Employees SET age=45, salary = 12000 WHERE id=9*



Если мы **не будем** использовать условие WHERE в нашем запросе, то все записи в указанном столбце будут обновлены. Пример:

*UPDATE Employees SET salary=1000*



В этом случае для всех сотрудников будет установлена ​​оклад в размере 1000.

**DELETE statement**

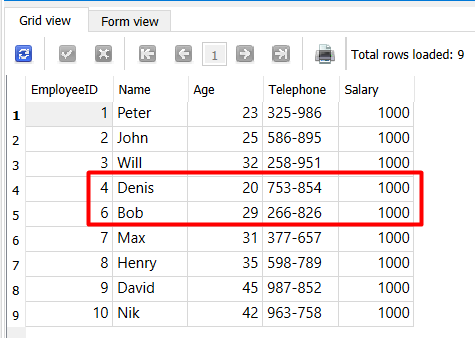
Оператор DELETE также является оператором языка манипулирования данными (DML), который позволяет нам манипулировать данными. Оператор DELETE удаляет записи из таблицы данных. Записи, подлежащие удалению, идентифицируются с использованием условия WHERE. Условие WHERE является необязательным. Если его не указать, то все данные в таблице будут удалены. Поэтому следует быть очень осторожным при использовании этого оператора.☺

Общий синтаксис запроса на удаление выглядит следующим образом:

*DELETE FROM Table\_Name WHERE condition*

Допустим, мы хотим удалить данные сотрудника с ID = 5. Для этого напишем запрос следующего вида:

*DELETE FROM Employees WHERE id=5*



Например, если мы хотим удалить всех сотрудников, чья зарплата больше 5000, то напишем такой запрос:

*DELETE FROM Employees WHERE salary>5000*