Конспект по теме "SQL как инструмент работы с данными"

Базы данных и таблицы

База данных — это хранилище структурированной информации. **Сущности** — группы объектов с общими характеристиками. **Объект** — отдельный представитель сущности.

Мы будем рассматривать реляционные базы данных. В них *сущности* — это *таблицы*, а *объекты — строки* таблиц.

Для работы с базами данных используется **СУБД** — система управления базами данных. Это комплекс программ, который позволяет создать базу данных, наполнить её новыми таблицами, отобразить содержимое, редактировать существующие таблицы. В этом курсе мы будем работать с одной из самых популярных СУБД — **PostgreSQL.**

Таблицы

Таблица — это совокупность строк и столбцов. Столбцы таблицы называются **поля.** В них обозначают характеристики объекта. У каждого поля есть уникальное имя и характерный тип данных. Строки таблицы называются **записи**. Каждая строка — информация об одном объекте. **Ячейка** — место пересечения строки и столбца.

Для того, чтобы записи в таблице базы данных определялись однозначно, используется особое поле — **первичный ключ.** Все значения в этом поле **уникальны**. Бывают в таблицах не только простые первичные ключи из одного поля, но и **составные первичные ключи.** Они состоят из нескольких полей.

Ваш первый SQL-запрос

SQL — язык программирования, предназначенный для управления данными в реляционной базе. Синтаксис SQL отличается от знакомого вам Python. Вот основные особенности:

1) Начало однострочного комментария обозначают двумя дефисами: `--`

```
-- однострочный комментарий на языке SQL
```

2) Многострочный комментарий заключают в `/*` косые черты со звёздочками `*/`:

```
/* многострочный
комментарий
называют
так,
```

```
потому что
в нём
много
строк */
```

3) Команды пишут заглавными буквами:

```
SELECT, WHERE, FROM
```

4) Каждый запрос заканчивается точкой с запятой `;` :

```
SELECT
  *
FROM
  название_таблицы;
-- Запрос на выборку всех данных из таблицы заканчивается ";"
SELECT
  *
FROM
  название_таблицы
WHERE
  название_столбца IN (1,7,9);
-- Запрос на выборку по условию тоже заканчивается ";"
```

5) Перенос строки делают после каждого ключевого слова:

```
SELECT
  название_столбца_1,
  название_столбца_2,
  название_столбца_3,
  название_столбца_4
FROM
  название_таблицы
WHERE
  название_столбца_1 = значение_1 AND
  название_столбца_2 = значение_2 AND
  название_столбца_4 = значение_3;
```

Чтобы выбрать данные из таблиц, нужно написать запрос. **Запрос** — это сформулированное в соответствии с синтаксисом SQL требование. В запросе объявляют, какие данные выбрать, и как именно их обработать.

Нужную выборку получают оператором **SELECT**. Синтаксис запроса с SELECT такой:

```
SELECT
  название_столбца_1,
  название_столбца_2,
  название_столбца_3 ...
FROM
  название_таблицы;
--Выбрать один столбец из таблицы
```

В запросе два ключевых слова: **SELECT** и **FROM**. **SELECT** выбирает нужные столбцы из таблицы базы данных. Чтобы выбрать из таблицы все столбцы, добавьте к оператору

`SELECT` символ `*` (звёздочка). **FROM** указывает, из какой таблицы брать данные.

Срезы данных в SQL

Начало условия, по которому отбираются данные, обозначают командой **WHERE**. Проверка на соответствие условию проходит в каждой строке таблицы. В условиях используются математические операторы сравнения:

```
SELECT
  название_столбца_1,
  название_столбца_2 --выбираем названия столбцов
FROM
  название_таблицы --указываем таблицу
WHERE
  условие; --определяем условие, по которому будем отбирать строки
```

Порядок операторов строго определён. Они должны идти так:

1) `SELECT`

2) `FROM`

3) `WHERE`

В SQL, как и в Python, есть логические операторы: `AND`, `OR`, `NOT`. Они позволяют делать выборку по нескольким условиям

```
SELECT
  *
FROM
  название_таблицы
WHERE
  условие_1 AND условие_2;
--Выбираются строки, которые соответствуют сразу обоим условиям

SELECT
  *
FROM
  название_таблицы
WHERE
  условие_1 OR условие_2;
--Выбираются строки, которые соответствуют хотя бы одному из условий

SELECT
  *
FROM
  название_таблицы
WHERE
  условие_1 AND NOT условие_2;
--Выбираются строки, которые соответствуют условию_1 и не соответствуют условию_2
```

Если нужно сделать выборку, условие которой - нахождение значения поля в определённом диапазоне, применяется конструкция **BETWEEN**. Границы диапазона в `BETWEEN` включены в результирующую выборку:

```
SELECT
  *
```

```
FROM
  название_таблицы
WHERE
  поле_1 BETWEEN значение_1 AND значение_2;
--Выбираются строки, в которых значение в поле_1 находится между значение_1 и значение_2, включительно
```

Когда нужно сделать выборку, в которой все значения поля находятся в определённом списке, используется оператор **IN**. После `IN` указывают список значений, которые нужно включить в результат:

```
SELECT
  *
FROM
  название_таблицы
WHERE
  название_столбца IN ('значение_1','значение_2','значение_3');
```

Если в списке должны быть числа, их указывают через запятую: `IN (3,7,9)`. Строки тоже через запятую, но в одинарных кавычках: `IN ('значение_1','значение_2','значение_3')`. Дату и время обозначают так: `IN ('чч-мм-ГГГГ','чч-мм-ГГГГ')`

Агрегирующие функции

Как и в Python, в SQL есть функции для подсчёта общего количества строк, суммы, среднего значения, максимума и минимума. Такие функции называют **агрегирующие.** Они собирают, или *агрегируют* все объекты группы, чтобы уже по ним вычислить нужные значения.

Пример формата запроса с агрегирующей функцией:

```
SELECT
  АГРЕГИРУЮЩАЯ_ФУНКЦИЯ(поле) AS here_you_are
--here_you_are - имя столбца, в котором сохранятся результаты работы функции
FROM
  TABLE
```

После вызова агрегирующей функции имя столбца выводится в неудобном виде. Чтобы этого избежать, применяют команду **AS**, а потом записывают новое имя столбца.

Функция **COUNT()** (англ. «подсчёт») возврщает количество строк в таблице:

```
SELECT
  COUNT(*) AS cnt
FROM
  table
```

В зависимости от задачи количество строк считают по-разному:

- **COUNT(*)** возвращает общее количество строк в таблице;
- **COUNT(column)** возвращает число строк в столбце `column`;

- **COUNT(DISTINCT column)** (англ. *distinct*, «отдельный, особый») возвращает количество *уникальных* строк в столбце `column`.

Функция **SUM(column)** возвращает сумму по столбцу `column`. При выполнении функции пропуски игнорируются.

AVG (column) возвращает среднее значение по столбцу `column`.

Минимальное и максимальное значения находят функциями **MIN()** и **MAX()**.

Изменение типов

Некоторые агрегирующие функции работают только с числовыми типами данных.

Данные в выборке выглядят как числа, но в базе данных хранятся как строки. В жизни такое встречается часто — обычно из-за ошибок проектирования базы данных.

Изменим тип данных столбца в самом SQL-запросе. Типы преобразуют конструкцией **CAST**:

```
CAST (название_столбца AS тип_данных)
```

Название_столбца — это поле, тип данных которого нужно преобразовать. *Тип_данных* — тип, в который данные нужно перевести. Есть и другая форма записи:

```
название_столбца :: тип_данных
```

В SQL используются следующие типы данных:

- Числовые типы данных
  - **integer** — целочисленный тип, аналогичный типу *int* в Python. В PostgreSQL диапазон целых чисел от -2147483648 до 2147483647.
  - **real** — число с плавающей точкой, как *float* в Python. Точность числа типа *real* до 6 десятичных разрядов.
- Строковые типы данных
  - **'Практикум'** — значение строкового типа, в SQL запросе его заключают в одинарные кавычки.
  - **varchar(n)** — строка переменной длины, где **n** — ограничение. Этот тип данных похож на *string* в Python, но в отличие от него ограничен по длине: в поле можно занести любую строку короче, чем **n** символов.
  - **text** — строка любой длины. Полный аналог *string* в *Python*.
- Дата и время. Любые вводимые значения даты или времени заключают в одинарные кавычки.
  - **timestamp** — дата и время. Этот тип аналогичен *datetime* в *Pandas*. В формате *timestamp* чаще всего хранят события, происходящие несколько раз за день.

Например, логи пользователей сайта.

- **date** — дата.

- **boolean** — логический тип данных. В PostgreSQL есть три варианта значений **TRUE** — «истина», **FALSE** — «ложь» и **NULL** — «неизвестно».