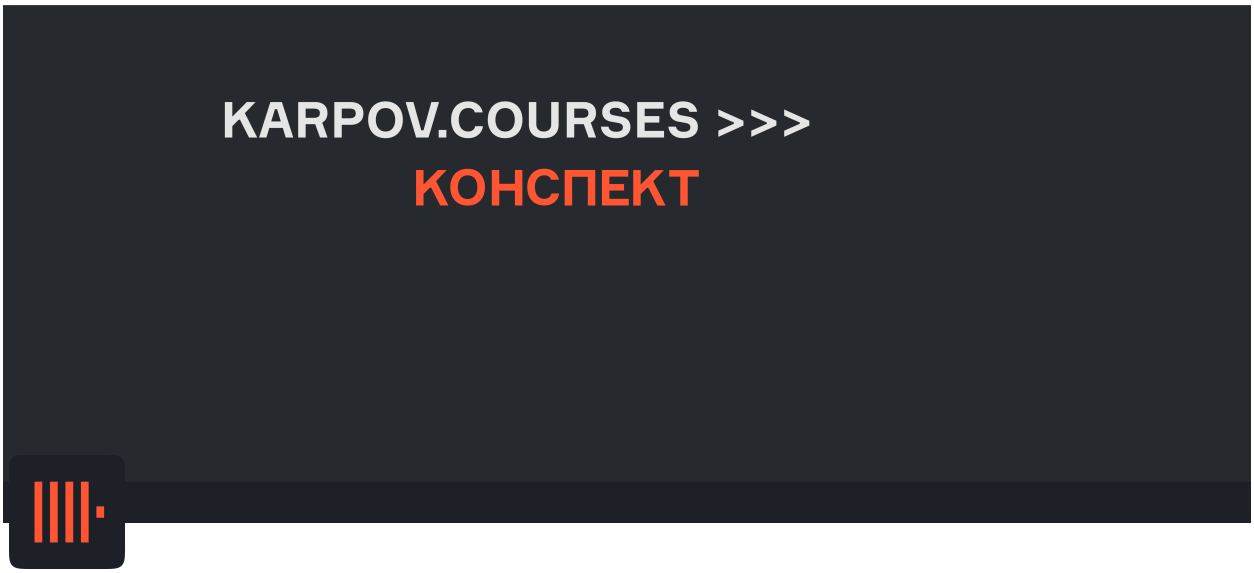

> Конспект > 1 урок > SQL

> Основные блоки SQL запросов

В этом уроке мы рассмотрим основные блоки запросов SQL.

1. Когда могут потребоваться базы данных?
2. Где хранят данные? + виды баз данных
3. Что такое SQL? ClickHouse?
4. Как подключиться к ClickHouse
5. `SELECT` + `FROM`
6. `AS` + `DISTINCT`
7. `WHERE`
8. `ORDER BY`
9. `LIMIT`
10. `MIN` + `MAX`
11. Порядок операторов

> Когда могут потребоваться базы данных?

Для каждой компании важно принимать правильные бизнес-решения. А именно – не на основе того, что "так чувствуется", а на основе данных.

В качестве примера возьмем интернет-магазин. В данном случае будут собираться данные:

- о визитах на сайт (напр. с помощью Яндекс.Метрики)
- данные онлайн платежей (уже другая система)
- данные по логистике (и еще одна)

Как же здесь принимать решения? Пока используется всего один источник данных, вполне можно это делать в Excel, или в каких-то других системах аналитики. Но когда источников уже много, нужно как-то корректно совмещать из них данные, причём так, чтобы не было потерь, чтобы вся логика была правильной. С простой системой аналитики это сделать сложно. Тогда и начинаются SQL и базы данных!

> Где хранят данные?

База данных – это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД). Задача сохранять данные, логировать и складывать – причина появления СУБД.

Иными словами, в чем разница между базой данных (БД) и системой управления базами данных (СУБД)? База данных – условно, просто "файл", который хранит данные. СУБД – то, что записывает данные, читает, модифицирует, удаляет и т.д., через нее происходит работа с БД.

Данные вместе с СУБД, а также приложения, которые с ними связаны, называются системой баз данных, или, для краткости, просто базой данных.

СУБД могут быть реляционными и NoSQL (not only SQL).

Реляционные базы данных

Элементы в реляционной базе данных организованы в виде набора таблиц со столбцами и строками. Технология реляционных баз данных обеспечивает наиболее эффективный и гибкий способ доступа к структурированной информации.

В реляционной базе данные разделены на таблицы, и одна таблица хранит однородные сущности, а связь между таблицами осуществляется некоторым общим ключом. Именно это свойство и отражено в названии, где слово реляционный происходит от англ. relation (отношения, зависимость).

Пример:

Клиент

cust_id	fname	lname
1	Джордж	Блейк
2	Сью	Смит

Счет

account_id	product_cd	cust_id	balance
103	CHK	1	$75.00
104	SAV	1	$250.00
105	CHK	2	$783.64
106	MM	2	$500.00
107	LOC	2	0

Тип счета

product_cd	name
CHK	Текущие расходы
SAV	Сбережения
MM	Денежный рынок
LOC	Кредитный лимит

Транзакция

txn_id	txn_type_cd	account_id	amount	date
978	DBT	103	$100.00	2004-01-22
979	CDT	103	$25.00	2004-02-05
980	DBT	104	$250.00	2004-03-09
981	DBT	105	$1000.00	2004-03-25
982	CDT	105	$138.50	2004-04-02
983	CDT	105	$77.86	2004-04-04
984	DBT	106	$500.00	2004-03-27

В целом, есть две версии происхождения названия:

1. реляционные – значит, что в базе есть таблицы, между ними есть какие-то связи (relations)
2. реляционные – потому что к ним применяется реляционная алгебра (операции про то, как работать с данными – объединить, пересечь, посчитать разность и т.д.)

Еще одним свойством является поддержка транзакций. **Транзакции** – операции обработки данных, которые переводят базу из одного состояния в другое. Допустим, есть банковская система, нужно перевести деньги одного пользователя на счет другого, и вы последовательно выполняете операции – вычесть у одного, зачислить другому.

Транзакции подчиняются аббревиатуре **ACID** – атомарность, согласованность, изолированность и стойкость.

- Атомарность – если мы что-то запустили (операции), то они либо все выполняются, либо все не выполняются.
- Согласованность – зафиксируются только допустимые результаты.
- Изолированность – если транзакции выполняются параллельно, то они не влияют друг на друга.
- Стойкость – если транзакция выполнена, то результат будет точно сохранен вне зависимости от проблем с оборудованием.

NoSQL

1. Ключ-значение – являются по сути словарем, позволяющим извлечь однозначное значение по ключу
2. Документо-ориентированные – похожи на ключ-значение, только значения с разметкой (XML, JSON), которая и образует "документ"
3. Столбцовые – данные хранятся по столбцам, а не по строкам
4. Графовые – вершины, ребра и их свойства

CAP теорема (в распределенных системах) – NoSQL БД может обладать двумя свойствами из трех:

- Согласованность (consistency)
- Доступность (availability)
- Устойчивость к разделению (partition tolerance)

> SQL – Structured Query Language

SQL (Structured Query Language, структурированный язык запросов) – это язык программирования, используемый в большинстве реляционных баз данных для запроса, обработки и определения данных, а также контроля доступа.

SQL является декларативным языком, то есть в нём не задается алгоритм решения задачи, а описывается, что нужно получить в качестве результата. В каком-то смысле SLQ похож на обычную человеческую речь. Можно сказать, что это язык для заказа нужных данных и операций над ними.

Существует множество реализаций SQL, каждая из которых имеет свои особенности, но в общем схожа со стандартом SQL. В данном курсе будет рассмотрена система управления базами данных ClickHouse (и немного PostgreSQL).

ClickHouse

ClickHouse – столбцовая система управления базами данных (СУБД) для онлайн обработки аналитических запросов (OLAP) (документация ClickHouse). Главные особенности:

- Столбцовая база данных для аналитического использовния (OLAP)
- Позволяет быстро читать большие данные
- Транзакции отсутствуют
- Данные добавляются, но не изменяются (изменения реализуется через инкремент)
- Нет оконных функций (пока что)

> Как подключиться к ClickHouse?

Для подключения к ClickHouse курса мы будем использовать интерфейс TABiX.

Подробнее о том, как работать с ClickHouse тут

1. Переходим на https://tabix.lab.karpov.courses
2. Заполняем поля:
   Name: ваше имя
   http://host:port: https://clickhouse.lab.karpov.courses
   Login: student
   Password: dpo_python_2020
3. Заходим!

Note: Нужен браузер основанный на chromium, иначе могут быть проблемы с подключением.О том, как поставить ClickHouse себе на компьютер можно почитать в документации.

Библиотека для интеграции в python – pandahouse.

> SELECT + FROM

`SELECT` – используется для указания того, какие из всех возможных столбцов таблицы попадут в результирующую таблицу.

Например, у нас есть таблица `sales` с именами менеджеров (`manager`) их идентификаторами (`id`) и размером дохода (`value`), который они принесли компании.

# id	Aa manager	# value
231	Гриценко	230000
275	Бердяев	370000
312	Курашов	150000
324	Савченко	98000

Чтобы в результат вывести всю таблицу нам понадобится еще один блок – `FROM`, который указывает из какой таблицы необходимо вывести указанные поля в запросе.

Конструкция запроса такова, что сначала необходимо поставить блок `SELECT`, далее перечислить все наименования полей, которые нужно вывести в результат, далее конструкция `FROM` и после блока `FROM` – имя таблицы, которая содержит нужные поля и их значения.

Для того, чтобы вывести в результат всю таблицу `sales` напишем запрос.

```
SELECT
    id, manager, value
FROM
    sales
```

Результат запроса:

# id	Aa manager	# value
231	Гриценко	230000
275	Бердяев	370000
312	Курашов	150000
324	Савченко	98000

Выбрать все поля можно не только их перечислением в запросе, но и с помощью символа :

```
SELECT *
FROM sales
```

> AS

При выводе результата вы можете переименовать имя поля. Сделать это можно добавлением алиаса `AS` перед “новым” именем после каждого элемента блока `SELECT`. Также алиас `AS` можно опустить, и указать новое имя поля без него.

```
SELECT
    id,
```

```
    manager AS names
FROM
    sales
```

или

```
SELECT
    id,
    manager names
FROM
    sales
```

Результат обоих запросов:

id	names
231	Гриценко
275	Бердяев
312	Курашов
324	Савченко

> DISTINCT

Иногда запрос может возвращать дублирующие строки в данных. Чтобы вывести **уникальные** значения поля или полей (т.е. уникальные строки) нужно поставить оператор `DISTINCT` после блока `SELECT` :

```
SELECT
    DISTINCT manager
FROM
    sales
```

> WHERE

Мы рассмотрели, как выводить полную таблицу со всеми строками и столбцами, но обычно извлекать все строки и столбцы не требуется. Чтобы

отфильтровывать строки, которые нам (не) интересны, используют блок `WHERE`.

Например, нам нужно посмотреть доход менеджера Савченко из таблицы `sales` – запрос будет выглядеть так:

```
SELECT *
FROM sales
WHERE manager = 'Савченко'
```

Результат запроса:

# id	Aa manager	# value
324	Савченко	98000

Несколько условий можно перечислить через `AND` или `OR`:

```
SELECT
    *
FROM
    sales
WHERE
    manager = 'Савченко'
    OR id = 275
```

# id	Aa manager	# value
275	Бердяев	370000
324	Савченко	98000

> ORDER BY

В результирующей таблице запрос возвращает данные в произвольном порядке. Если нужно упорядочить вывод запроса определенным образом используйте блок `ORDER BY`. Чтобы отсортировать по убыванию или по

возрастанию, укажите `DESC` или `ASC` после наименования поля, по которому требуется сортировка.

- `DESC` – сортировка по убыванию
- `ASC` – по возрастанию

Если тип сортировки не задан, то **по умолчанию** будет использована сортировка **по возрастанию** (ASC).

Отсортируем таблицу `sales` по убыванию дохода, который менеджеры принесли компании:

```
SELECT
    manager,
    value
FROM
    sales
ORDER BY
    value DESC
```

Результат запроса:

# id	Aa manager	# value
275	Бердяев	370000
231	Гриценко	230000
312	Курашов	150000
324	Савченко	98000

> LIMIT

Теперь давайте представим, что нам нужно получить топ N результатов. Для этого нам потребуется оператор `LIMIT`. Ему мы передаем либо один параметр `N`, если нужно извлечь `N` строк с начала таблицы с данными, либо два параметра – первый устанавливает смещение от первой строки, то есть сколько строк нужно пропустить, а второй указывает на количество извлекаемых строк.

Давайте выведем топ 3 менеджера по доходу, который они принесли компании. У нас уже есть готовый запрос из предыдущего примера – доработаем его:

```
SELECT
    manager,
    value
FROM
    sales
ORDER BY
    value DESC
LIMIT 3
```

Результат запроса:

# id	Aa manager	# value
275	Бердяев	370000
231	Гриценко	230000
312	Курашов	150000

Не забывайте использовать `LIMIT`, чтобы случайно не вывести огромную табличку и что-нибудь не уронить :)

> MIN + MAX

`MIN()` – показывает минимальное значение `MAX()` – показывает максимальное значение

Например, попробуем выбрать наименьшее значение `value` из следующей таблицы:

# id	Aa manager	# value
231	Гриценко	230000
275	Бердяев	370000
312	Курашов	150000
324	Савченко	98000

```
SELECT
    MIN(value) AS min_value
FROM
    sales
```

Результат:

Aa Title	# min_value
Untitled	98000

Более подробно агрегатные функции разберем в следующей лекции :) stay tuned!

> Порядок операторов

Обычно операторы используются в следующем порядке:

```
SELECT -- перечисление полей для результирующей таблицы
    BuyDate,
    COUNT(*) AS rows
FROM -- источник данных
    default.checks
WHERE -- фильтрация данных
    BuyDate != '2019-03-08'
GROUP BY -- группировка данных
    BuyDate
HAVING -- фильтрация данных, аналогично WHERE, при использовании GROUP BY
    COUNT(*) > 215000
ORDER BY -- сортировка результирующей таблицы
    rows DESC
LIMIT 100 -- ограничение на кол-во строк результирующей таблицы
```