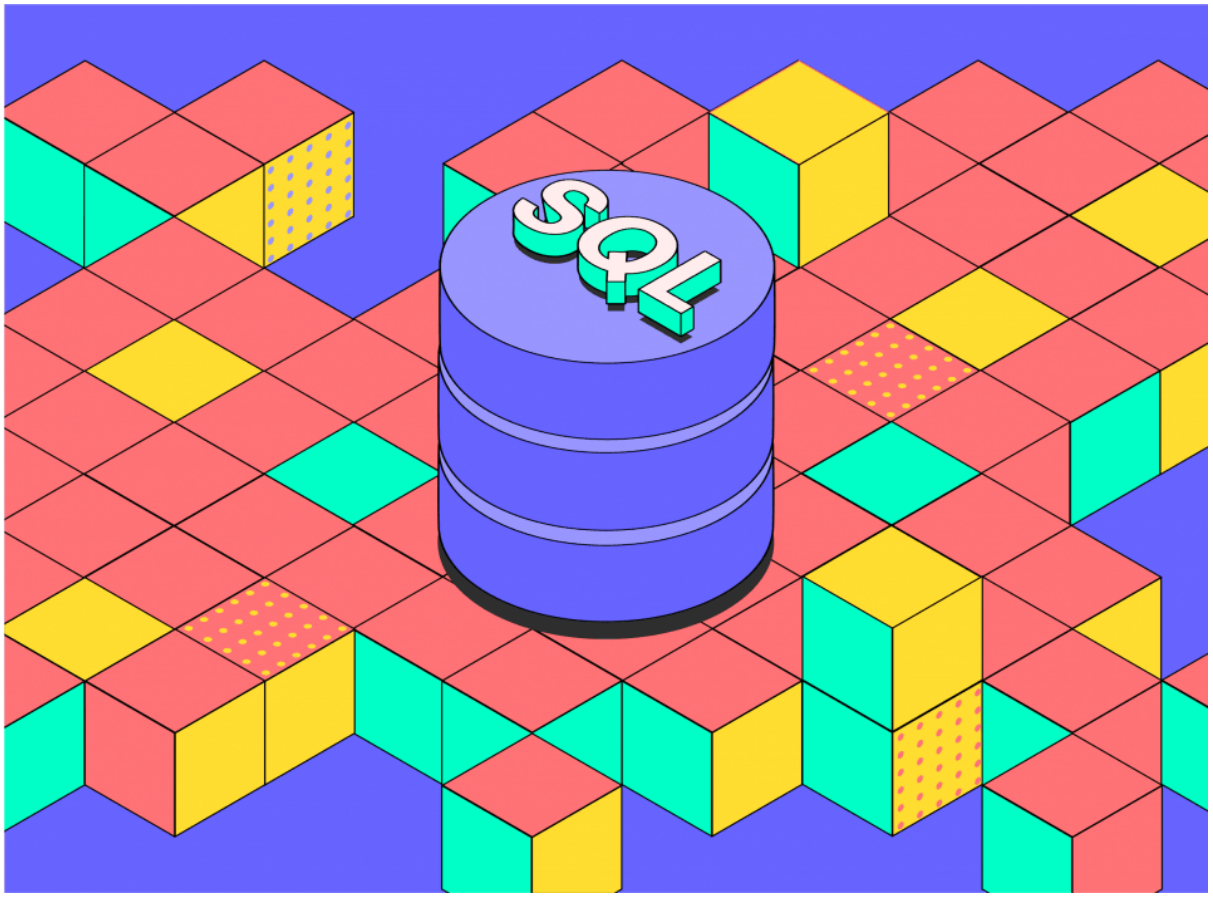
SQL note’s
by Mike Zigberman
russian edition



Введение в базы данных

Работа с данными — это большая часть работы с любым проектом. Информация о пользователях, тексты и фотографии на сайте, информация о клиентах банка — всё это данные, с которыми работает программист. Более того, с данных и начинается проект.
Для хранения и обработки информации созданы **базы данных.**

Управление данными происходит посредством **систем управления базами данных** или **СУБД**.

Базы данных различаются по назначению и типам операций, под которые они оптимизированы. Реляционные базы данных подходят для хранения связанной информации (от англ. *relation* — «отношение, зависимость, связь»). Есть базы, оптимизированные для поиска по текстам с учетом человеческого языка. Базы графов позволяют анализировать взаимосвязи между объектами, например, между людьми в социальных сетях или изучать банковские операции, чтобы выявить мошенников. Некоторые базы позволяют хранить информацию в оперативной памяти сервера, это удобно, когда скорость обмена данными критически важна.

В работе веб-приложений чаще всего применяются реляционные базы данных, и в дальнейшем под словами «база данных» будут подразумеваться именно они.

Таблицы

В базах данных информация хранится в таблицах.
Вот база из трёх таблиц, в которых отдельно хранится информация о литературных произведениях, об авторах и об их супругах:

author

id	name	birth_year
1	Антон Чехов	1860
2	Владимир Набоков	1899
3	Лев Толстой	1828
4	Насон Грядущий	3019
5	Александр Грибоедов	1795
6	Александр Островский	1823
7	Марина Цветаева	1892
8	Александр Герцен	1812
9	Фёдор Достоевский	1821
10	Николай Некрасов	1821
11	Николай Чернышевский	1828
12	Софья Ковалевская	1850

book

id	author_id	title	year
1	5	Горе от ума	1824
2	8	Кто виноват?	1846
3	7	Волшебный фонарь	1912
4	6	Утро молодого человека	1850
5	6	Бедная невеста	1851
6	8	Былое и думы	1852
7	11	Что делать?	1863
8	3	Заражённое семейство	1864
9	3	Нигилист	1866
10	10	Кому на Руси жить хорошо	1877
11	7	Вечерний альбом	1910
12	3	Власть тьмы, или Коготок увяз, всей птичке пропасть	1886
13	1	О вреде табака	1886
14	1	Иванов	1887
15	9	Крокодил	1865
16	9	Вечный муж	1870
17	1	Трагик поневоле	1889
18	12	Воспоминания детства	1890
19	9	Записки из подполья	1864
20	2	Защита Лужина	1930
21	12	Воспоминания о Джордже Эллиоте	1886
22	2	Подвиг	1932

spouse

id	name	author_id	wed_year
1	Мария Бахметьева	6	1869
2	Фёкла Викторова	10	1877
3	Сергей Эфрон	7	1912
4	Ольга Васильева	11	1853
5	Мария Исаева	9	1857
6	Наталья Захарьина	8	1838
7	Вера Слоним	2	1925
8	Софья Берс	3	1862
9	Нино Чавчавадзе	5	1828
10	Ольга Книппер	1	1901
11	Владимир Ковалевский	12	1868
12	Анна Сниткина	9	1867

Каждая строка таблицы — это отдельная **запись**. Каждая запись таблицы содержит **поля** — ячейки таблицы. Поля называют по имени колонки: например, записи в таблице *spouse* содержит поля *id*, *name, author_id* и *wed_year*

Когда в 3044 году Насон Грядущий женится — в таблице *spouse* появится ещё одна **запись**, строка таблицы. Она будет содержать поля с такими значениями:

```
id = 13
name = "Агриппина Будущая"
author_id = 4 -- это id Насона Грядущего в таблице author
wed_year = 3044
```

При создании таблиц обязательно указывается перечень полей и значения какого типа эти поля могут хранить.
В таблице со списком произведений созданы поля `id`, `author_id`, `title` и `year`. Поля `id`, `author_id` и `year` хранят данные типа «целое число» `int`, а поле `title` хранит строковые данные `str`.

Примеры самых часто используемых типов:

- **Целое число.** Ради экономии имеет смысл выбирать тип данных, который занимает меньше места, особенно если известно, что данные не могут превышать пороговое значение. Например, предельный возраст человека пока что ограничен, поэтому можно сэкономить, если в колонке *age* хранить только числа в диапазоне от 0 до 255.
- **Число с плавающей точкой.** Отдельный тип для дробных числовых величин. Подходит, например, для хранения веса товара или расстояний в световых годах.
- **Одиночный текстовый символ.** Этот формат может применяться для хранения однобуквенного индекса товара, обозначения корпуса дома или школьного класса.
- **Строка.** Это тип для хранения любого набора символов. Текст статьи или список покупок в домашнем todo — всё это «строка».
- **Монетарный тип** служит для хранения информации о денежных суммах. Обычно банки имеют отдельные правила округления чисел монетарных типов.
- **Дата и время**. В базах данных обычно есть несколько специализированных типов для хранения дат, времени или временных интервалов.

Помимо базовых типов могут применяться и специальные, например — для хранения картографической информации, адресов компьютеров в сети или документов в формате JSON или XML.

При сохранении информации в БД данные проверяются на соответствие требуемому типу. Попробуйте записать строковое значение в поле для чисел — и получите сообщение об ошибке.

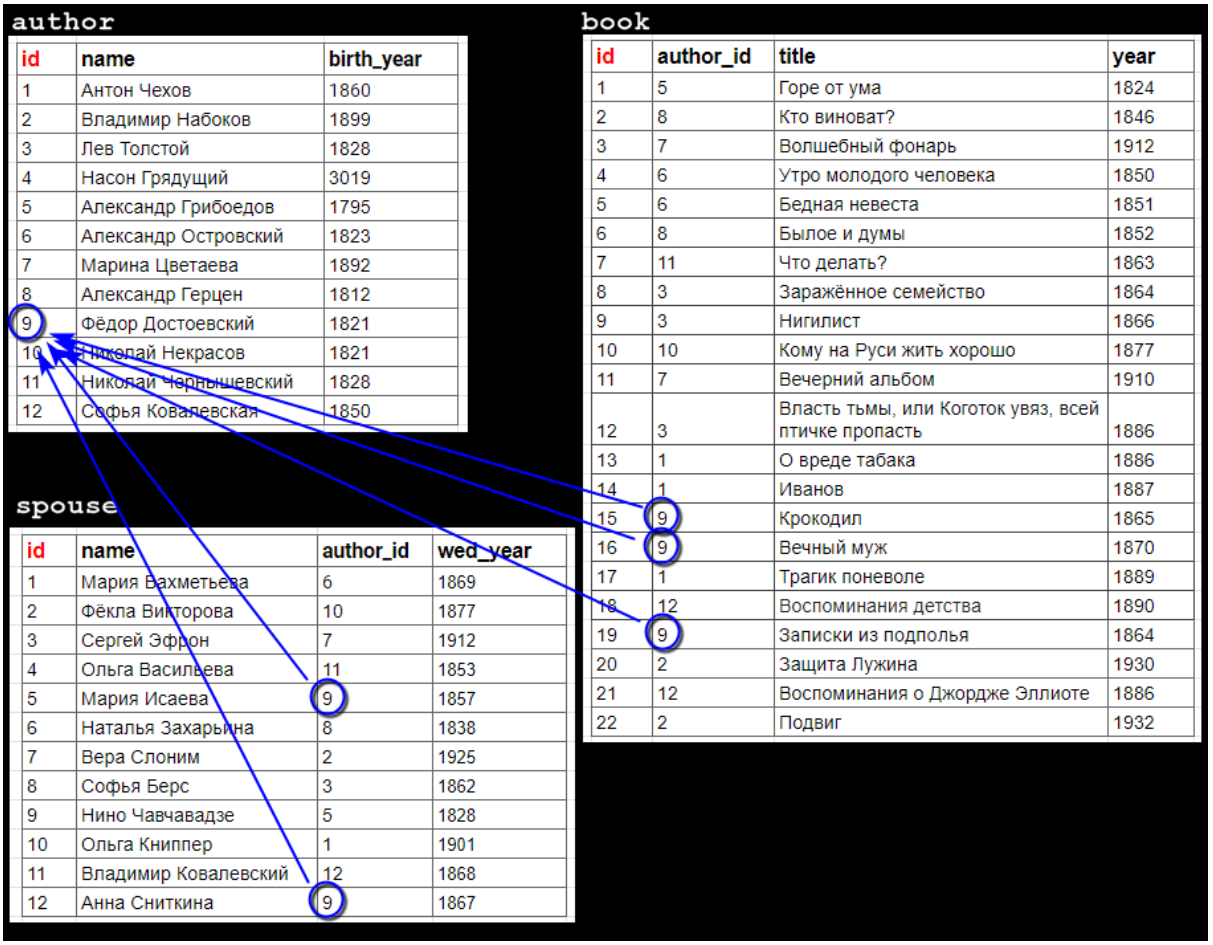
Записи в разных таблицах могут быть связаны между собой. В нашем примере данные об авторах хранятся отдельно от информации о произведениях, которые они написали. Для того, чтобы определить авторство, между таблицами выстроены связи. Вот откуда взялось название «*реляционная* база данных».

Данные столбца одной таблицы могут быть ссылками на записи в другой таблице. И тут начинается самое интересное: правильно

подобрав условия и ограничения, можно получить возможность эффективно обрабатывать огромные массивы информации.

Ещё раз посмотрите на картинку: в таблицах *book* и *spouse* есть колонка `author_id`, в этой колонке указан ID автора, с которым связана та или иная запись. В результате средствами СУБД мы всегда можем получить и обработать связанные данные.

Например, наша база данных позволяет узнать, на ком был женат автор произведения «Крокодил», хотя эти данные лежат в разных таблицах.



SQL

Для работы с реляционными базами данных придуман специальный структурированный язык запросов — Structured Query Language (сокращенно SQL). Иногда в шутку его называют *Simple Query Language* — «простой язык запросов»: SQL — это одна из самых

сложных технологий, с которыми программистам приходится постоянно работать.

Любое взаимодействие с базой данных происходит посредством запросов на языке SQL. Запрос — это команда: «база данных, сделай то-то и то-то».

Например:

- Покупатель сделал заказ в интернет-магазине — к БД отправляется SQL-запрос: «в таблицу *buyer* в новую строку записать данные покупателя; в таблице *basket* отметить заказанные товары».
- Пользователь запросил подборку статей по тегу "What is SQL" — к БД отправляется SQL-запрос: «выбрать из таблицы *articles* те статьи, у которых в поле *tag* есть запись "What is SQL" и вернуть для каждой статьи содержимое полей *title*, *date*, *main_image*, *description, tag*; для каждой статьи запросить имя автора из таблицы *author*; статьи отсортировать по полю *date*, от новых к старым».
- В социальной сети зарегистрировался новый пользователь — к БД отправляется SQL-запрос: «в таблицу *users* добавить новую строку с данными пользователя».

Запрос — это текст, команда, которая отправляется к базе данных.

Пример запроса:

```
SELECT * FROM author;
```

Пример ответа:

```
id     name                   birth_year
-----  ---------------------  ------------
1      Человек без селезенки  1860
2      Владимир Набоков       1899
3      Лев Толстой            1828
4      Насон Грядущий         3019
```

Можно отправить запросы на чтение, изменение и добавление данных. Также существуют служебные запросы, например — запрос на создание новой таблицы.

Каждый запрос обязательно заканчивается точкой с запятой `;`.
Ключевые слова запросов пишут заглавными буквами. Можно писать и строчными, но обычно так не делают: читаемость кода ухудшается.

Создание таблицы

Синтаксис для создания таблицы выглядит так:

```
CREATE TABLE <имя таблицы> (
    <имя столбца1> <тип столбца> [дополнительные условия],
    <имя столбца2> <тип столбца> [дополнительные условия],
);
```

Вот настоящий запрос на создание таблицы:

```
-- СОЗДАТЬ ТАБЛИЦУ с названием author
CREATE TABLE author(
    id    INTEGER PRIMARY KEY,
    -- создать колонку с названием id, в ней будут ЦЕЛЫЕ ЧИСЛА,
    -- в этой колонке будут храниться УНИКАЛЬНЫЕ КЛЮЧИ записей
    name  TEXT NOT NULL,
    -- создать колонку с названием name, в ней будет ТЕКСТ
    -- и НЕ МОЖЕТ БЫТЬ ПУСТОЙ
    birth_year INTEGER
    -- создать колонку с названием birth_year, в ней будут ЦЕЛЫЕ ЧИСЛА
);
```

Эта команда создаёт новую таблицу `author` со столбцами `id`, `name` и `birth_year`. У `id` и `birth_year` тип допустимых значений — `integer`, целое число.

Для столбца `id` указано условие: это `primary key`, первичный ключ.

Это значит, что каждой записи в таблице будет присвоен уникальный номер, по которому потом можно будет её найти.
Уникальный номер налогоплательщика, номер паспорта, заводской VIN-код автомобиля, государственный номер автомобиля — всё это уникальные, не повторяющиеся идентификаторы. Такой же уникальный идентификатор есть и у каждой записи в БД.

Особые правила для числовых первичных ключей:

- Если при добавлении новой записи в таблицу не указать первичный ключ, то база сама присвоит его.
- Если удалить запись, то её первичный ключ не будет повторно использован при автоматическом присвоении ключей. Но можно добавить новую запись с ключом удалённой записи, если при создании записи указать ключ явно.

CRUD создать, прочитать, обновить, удалить

После создания таблицы в нее можно добавлять записи. Проще всего представлять себе, что каждая новая запись — это строка, а поля — это ячейки таблицы.

Операции с данным часто именуют сокращением **CRUD**:

- **C**reate — создать запись
- **R**ead — прочитать данные
- **U**pdate — обновить запись
- **D**elete — удалить запись

Добавление записей: INSERT

Синтаксис добавления новой записи в таблицу выглядит так:

```
INSERT INTO <имя таблицы> (<имя столбца>[, <имя столбца2>, ...])

VALUES (<значение>[, <значение2>, ...]);
```

Для того чтобы добавить несколько записей — отправим несколько запросов `INSERT` («вставить»):

```
INSERT INTO author (name, birth_year) VALUES ('Человек без селезёнки',
1860);

-- ВСТАВИТЬ строку В ТАБЛИЦУ author

-- заполнить поля (name, birth_year) ЗНАЧЕНИЯМИ 'Человек без
селезёнки', 1860 соответственно
```

```
INSERT INTO author (name, birth_year) VALUES ('Владимир Набоков',
1899);

INSERT INTO author (name, birth_year) VALUES ('Лев Толстой', 1828);

INSERT INTO author (name, birth_year) VALUES ('Насон Грядущий', 3019);

INSERT INTO author (name, birth_year) VALUES ('Юрий Олеша', 1899);

INSERT INTO author (name, birth_year) VALUES ('Николай Чернышевский',
1828);

INSERT INTO author (name, birth_year) VALUES ('Андрей Платонов', 1899);
```

Пусть база сама присваивает номера уникальных ключей, для поля id передавать значения мы не будем.

Чтение записей: SELECT

Синтаксис запросов на чтение:

```
SELECT

    <перечень столбцов>

FROM

    <перечень таблиц>

WHERE

    <условия>;
```

В ответ на запрос **SELECT** данные возвращаются в структурированном табличном виде. Чтобы не путаться в понятиях «таблица БД» и «таблица с ответом» будем называть возвращаемые данные **«результирующая выборка»**.

Прочитаем записи которые мы добавили в таблицу:

```
-- ВЫБРАТЬ поля id, name, birth_year

SELECT

    id,

    name,
```

```
    birth_year

-- ИЗ ТАБЛИЦЫ author

FROM

    author;
```

Если нам нужно получить все поля, то вместо их перечисления можно использовать символ `*`:

```
SELECT * FROM author;
```

Этот запрос вернёт результат:

```
-- ВЫБРАТЬ данные из всех колонок ИЗ ТАБЛИЦЫ author

SELECT * FROM author;

-- Ответ (вот ты какая, «результирующая выборка»!):

id      name                    birth_year

-----   --------------------    ------------

1       Человек без селезёнки   1860

2       Владимир Набоков        1899

3       Лев Толстой             1828

4       Насон Грядущий          3019

5       Юрий Олеша              1899

6       Николай Чернышевский    1828

7       Андрей Платонов         1899
```

Чтобы показать только записи, содержащие в определённом поле уникальное значение — применяют оператор **DISTINCT:**

```
-- ВЫБРАТЬ поля id, name, но только записи с уникальным значением поля
birth_year

SELECT

    id,
```

```
    name,
    DISTINCT birth_year
-- ИЗ ТАБЛИЦЫ author
FROM
    author;
-- Ответ (только уникальные годы рождения):
id      name                    birth_year
-----   --------------------    ------------
1       Человек без селезёнки   1860
2       Владимир Набоков        1899
3       Лев Толстой             1828
4       Насон Грядущий          3019
```

Если мы хотим получить какую-то определённую запись, то в запросе надо указать условия. Для этого существует команда **WHERE**:

```
-- ВЫБРАТЬ данные из всех колонок ИЗ таблицы author
-- только в той строке, У КОТОРОЙ в поле id указано значение 3
SELECT * FROM author WHERE id=3;
-- Ответ:
id      name                    birth_year
-----   --------------------    ------------
3       Лев Толстой             1828
```

Изменение записей: UPDATE

Для обновления данных в существующей записи предназначена команда **UPDATE**

```
UPDATE <таблица>
```

```
SET <столбец> = <значение>[, <столбец2> = <значение>, ...]

WHERE <условие>;
```

Чтобы изменить запись с *id = 1*, можно использовать такой запрос:

```
-- ОБНОВИТЬ таблицу author

-- и ЗАПИСАТЬ в поле name значение 'Антон Чехов'

-- только в той строке, У КОТОРОЙ в поле id указано значение 1

UPDATE author SET name = 'Антон Чехов' WHERE id=1;
```

Если не указать условие и отправить запрос `UPDATE author SET name = 'Антон Чехов'`, то изменится значение `name` **абсолютно всех** записей в таблице: в БД окажутся семь Антонов Чеховых с разными годами рождения.
Посмотрим, как изменились данные в таблице:

```
-- ОБНОВИТЬ таблицу author и ЗАПИСАТЬ в поле name

-- значение 'Антон Чехов' в той строке, У КОТОРОЙ в поле id указано 1

UPDATE author SET name = 'Антон Чехов' WHERE id=1;

-- ВЫБРАТЬ данные из всех колонок ИЗ таблицы author в той строке,

-- У КОТОРОЙ в поле id указано 1

SELECT * FROM author WHERE id=1;

-- Ответ:

id      name                    birth_year

-----   --------------------    ------------

1       Антон Чехов             1860

-- Порядок, вернули классику настоящее имя
```

Удаление записей: DELETE

Синтаксис запросов для удаления:

```
DELETE FROM <таблица> WHERE <условие>;
```

Как и в случае с **UPDATE**, условие **WHERE** имеет очень большое значение: если его не указать, то запрос может удалить вообще все записи в таблице.

Удалим из базы автора, который пока что не входит в список лучших русскоязычных авторов мира и проверим, что получилось.

```
-- УДАЛИТЬ ИЗ таблицы author запись, У КОТОРОЙ в поле id указано значение 4

DELETE FROM author WHERE id=4;

-- ВЫБРАТЬ данные из всех колонок ИЗ таблицы author

SELECT id, name FROM author;

-- Ответ:

id      name

-----  --------------------

1       Антон Чехов

2       Владимир Набоков

3       Лев Толстой

5       Юрий Олеша

6       Николай Чернышевский

7       Андрей Платонов

--  Насон Грядущий has left the building
```

Условия запроса: WHERE

WHERE позволяет использовать множество условий одновременно. Для объединения сразу нескольких условий используются операторы **AND**, **OR** или **NOT**, с этими логическими операторами вы уже работали в теме про ветвления.

Пример:

```
-- ВЫБРАТЬ поля id и name ИЗ ТАБЛИЦЫ author

-- в записях, У КОТОРЫХ поле birth_year=1860 ИЛИ поле birth_year=1899

SELECT id, name FROM author WHERE birth_year=1860 OR birth_year=1899;

-- Ответ:

id           name

----------  ----------------

1            Антон Чехов

2            Владимир Набоков

5            Юрий Олеша

7            Андрей Платонов
```

Такой же запрос можно задать через условие **вхождения значения во множество IN**:

```
-- ВЫБРАТЬ поля id и name ИЗ ТАБЛИЦЫ author в записях,

-- У КОТОРЫХ значение поля birth_year совпадает хотя бы с одним
значением В СПИСКЕ (1860,1899)

SELECT id, name FROM author WHERE birth_year IN (1860,1899);

-- Ответ получили такой же:

id           name

----------  ----------------

1            Антон Чехов

2            Владимир Набоков
```

```
5            Юрий Олеша

7            Андрей Платонов
```

Со скобками можно задать более сложные условия:

```
SELECT * FROM author WHERE (id < 4 OR id > 5) AND birth_year = 1899;

-- Ответ:

id           name                  birth_year

----------  ----------------      -----------

2            Владимир Набоков      1899

7            Андрей Платонов       1899
```

Операторы, которые вам понадобятся при составлении запросов:

- `=` это оператор **сравнения**, а не присваивания.
- `>` , `<` — больше и меньше, `>=` , `<=` — больше или равно и меньше или равно.
- `<>` — не равно. В некоторых базах данных применяется `!=` в качестве оператора неравенства.
- `BETWEEN` — «между», для проверки значения в диапазоне. Например: `birth_year BETWEEN 1850 AND 1900`.
- `IN` — вхождение в список. Пример использования `city IN ('Москва', 'Днепр')`.
- `LIKE`, `ILIKE` — поиск строки по шаблону и поиск строки по шаблону без учёта регистра. Пример: `city LIKE 'Днепр%'`, символ `%` заменяет любой набор символов: такой маске будут соответствовать значения поля *Днепр*оГЭС, *Днепр* или *Днепр*овский (а вот «днепровский», с маленькой буквы, не сработает. В этой ситуации нужен оператор `ILIKE`). Базы данных могут поддерживать и другие маски.

Базы данных могут иметь дополнительные функции или операторы для преобразования строк или поиска с учетом морфологии языка.

Агрегирующие функции. Функция COUNT

В SQL есть функции для подсчета общего количества строк, суммы, среднего значения, максимума и минимума. Такие функции называют **агрегирующие.**

Они собирают или *агрегируют* записи (строки таблиц) по заданным условиям и затем проводят над найденными записями какие-то операции.

Пример формата запроса с агрегирующей функцией:

```
SELECT
    АГРЕГИРУЮЩАЯ_ФУНКЦИЯ(поле) AS result_name
    -- result_name - имя столбца результирующей выборки
FROM
    ИмяТаблицы;
```

Оператор SELECT возвращает результат в виде таблицы. В случае с агрегирующей функцией таблица будет состоять из одной колонки. Имя этой колонки задаётся командой **AS** (англ. «как»).

Например, агрегирующая функция **COUNT()** (англ. «подсчёт») возвращает количество строк в таблице:

```
SELECT
    -- СОСЧИТАТЬ все строки и вернуть результат В СТОЛБЦЕ ПО ИМЕНИ cnt
    COUNT(*) AS cnt
FROM
    -- ИЗ ТАБЛИЦЫ author
    author;
-- Ответ вернёт результат: "нашлось шесть строк" (Насона Грядущего мы удалили)
cnt
-----
6
```

Для того, чтобы найти дату рождения самого старшего автора (запись с наименьшим значением в поле `birth_year`) применим агрегирующую функцию **MIN**:

```
SELECT
    -- НАЙТИ НАИМЕНЬШЕЕ ЗНАЧЕНИЕ в колонке birth_year
    -- и вернуть результат В СТОЛБЦЕ ПО ИМЕНИ min_year
    MIN(birth_year) AS min_year
FROM
    author;
-- Ответ (обратите внимание: возвращается результат подсчёта, а не запись):
min_year
----------
1828
```

Существуют агрегирующие функции для подсчета числовых значений:

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

Функция **SUM(column)** возвращает сумму по столбцу `column`.

В практических заданиях вы будете работать с базой данных товаров и продаж. Информация хранится в таблицах `products_data_all` и `transactions`.

Связи между таблицами

Запрос может быть обращен одновременно к нескольким таблицам базы данных. Для этого:

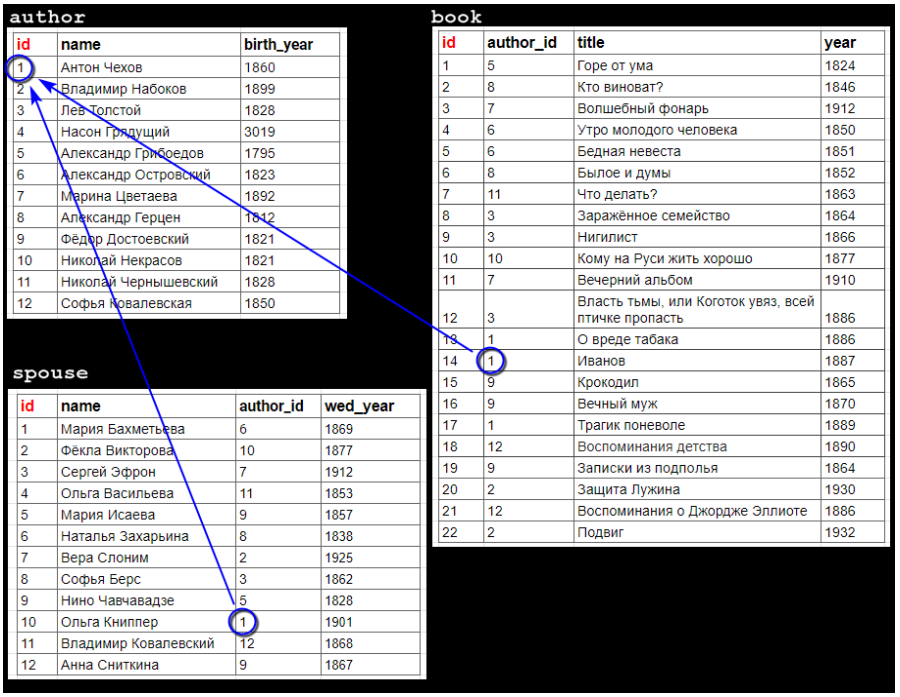
- В поле FROM указывают, данные из каких таблиц надо получить.
- В поле SELECT перечисляют имена столбцов, которые попадут в результирующую выборку. Если в разных таблицах имена столбцов повторяются, то их надо указывать по правилу `ИмяТаблицы.ИмяСтолбца`.
- В поле WHERE указывают условия получения данных из таблиц.

Синтаксис запроса к нескольким таблицам выглядит так:

```
SELECT
    -- Столбцы, данные из которых мы хотим получить в ответе на запрос.
    -- Для удобства можно задать новое имя для столбца командой AS
    ИмяТаблицы1.ИмяСтолбца1 AS firstField,
    ИмяТаблицы2.ИмяСтолбца1 AS secondField,
    ...
FROM
    -- Таблицы, из которых запрашиваем данные:
    ИмяТаблицы1,
    ИмяТаблицы2,
    ...
WHERE
    -- Условия используют ту же нотацию: ИмяТаблицы.ИмяСтолбца
    -- Указываем, какими полями связаны записи в запрошенных таблицах
    ИмяТаблицы1.ИмяСтолбцаN = ИмяТаблицы2.ИмяСтолбцаM
```

Наиболее частый способ для связи между таблицами — это связь через поля с первичным ключом.

Каждая запись в таблицах *book* и *spouse* ссылается на значение в колонке `id` таблицы *author*. По этой связи несложно определить, кто написал пьесу «Ива́нов» или чьей женой была Ольга Книппер.



author

id	name	birth_year
1	Антон Чехов	1860
2	Владимир Набоков	1899
3	Лев Толстой	1828
4	Насон Грядущий	3019
5	Александр Грибоедов	1795
6	Александр Островский	1823
7	Марина Цветаева	1892
8	Александр Герцен	1812
9	Фёдор Достоевский	1821
10	Николай Некрасов	1821
11	Николай Чернышевский	1828
12	Софья Ковалевская	1850

book

id	author_id	title	year
1	5	Горе от ума	1824
2	8	Кто виноват?	1846
3	7	Волшебный фонарь	1912
4	6	Утро молодого человека	1850
5	6	Бедная невеста	1851
6	8	Былое и думы	1852
7	11	Что делать?	1863
8	3	Заражённое семейство	1864
9	3	Нигилист	1866
10	10	Кому на Руси жить хорошо	1877
11	7	Вечерний альбом	1910
12	3	Власть тьмы, или Коготок увяз, всей птичке пропасть	1886
13	1	О вреде табака	1886
14	1	Иванов	1887
15	9	Крокодил	1865
16	9	Вечный муж	1870
17	1	Трагик поневоле	1889
18	12	Воспоминания детства	1890
19	9	Записки из подполья	1864
20	2	Защита Лужина	1930
21	12	Воспоминания о Джордже Эллиоте	1886
22	2	Подвиг	1932

spouse

id	name	author_id	wed_year
1	Мария Бахметьева	6	1869
2	Фёкла Викторова	10	1877
3	Сергей Эфрон	7	1912
4	Ольга Васильева	11	1853
5	Мария Исаева	9	1857
6	Наталья Захарьина	8	1838
7	Вера Слоним	2	1925
8	Софья Берс	3	1862
9	Нино Чавчавадзе	5	1828
10	Ольга Книппер	1	1901
11	Владимир Ковалевский	12	1868
12	Анна Сниткина	9	1867

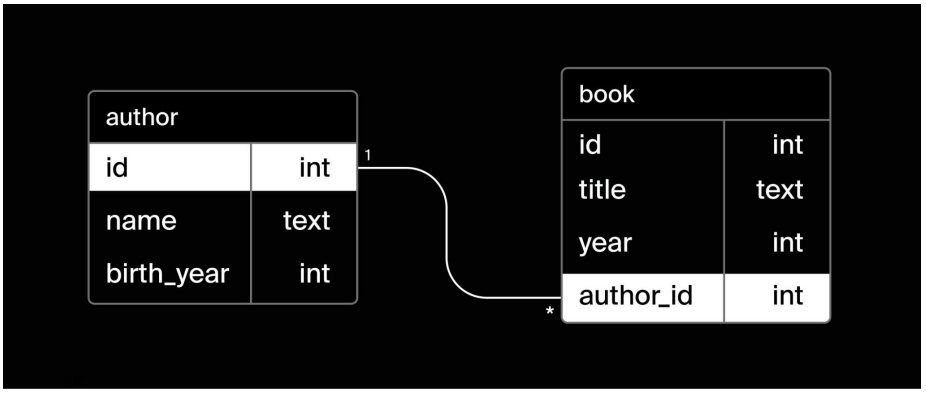
При создании таблицы в базе данных мы создавали поле, обозначенное как «первичный ключ», `PRIMARY KEY`. Сейчас этот ключ понадобится, так что повторим его свойства:

- Первичные ключи уникальны, в колонке со свойством PRIMARY KEY значения не могут повторяться.
- При создании новых записей базы данных могут сами генерировать значения для первичных ключей.
- В некоторых СУБД можно проверять, есть ли запись в таблице, на которую ссылается запись из другой таблицы.

Продолжим работать с базой русской литературы. Таблица *author* содержит имена и годы рождения авторов. Таблица *book* содержит название и год написания книги, а также ID автора из таблицы *author*.

```
-- создаём таблицу author и определяем, что в ней будут
-- колонки id, name и birth_year
CREATE TABLE author(
    -- в таблице author в колонке id содержатся целые числа,
    -- в этой колонке каждая ячейка содержит уникальное значение
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    birth_year INTEGER
);

-- создаём таблицу book
CREATE TABLE book (
    id INTEGER PRIMARY KEY,
    title VARCHAR,
    year INTEGER,
    author_id INTEGER,
    -- Это специальный синтаксис, который связывает
    -- записи с таблицей авторов
    FOREIGN KEY(author_id) REFERENCES author(id)
);
```

Графическое представление связи между таблицами:



Заполним таблицу авторов и таблицу книг:

author

id	name	birth_year
1	Антон Чехов	1860
2	Владимир Набоков	1899
3	Лев Толстой	1828
4	Насон Грядущий	3019
5	Александр Грибоедов	1795
6	Александр Островский	1823
7	Марина Цветаева	1892
8	Александр Герцен	1812
9	Фёдор Достоевский	1821
10	Николай Некрасов	1821
11	Николай Чернышевский	1828
12	Софья Ковалевская	1850

book

id	author_id	title	year
1	5	Горе от ума	1824
2	8	Кто виноват?	1846
3	7	Волшебный фонарь	1912
4	6	Утро молодого человека	1850
5	6	Бедная невеста	1851
6	8	Былое и думы	1852
7	11	Что делать?	1863
8	3	Заражённое семейство	1864
9	3	Нигилист	1866
10	10	Кому на Руси жить хорошо	1877
11	7	Вечерний альбом	1910
12	3	Власть тьмы, или Коготок увяз, всей птичке пропасть	1886
13	1	О вреде табака	1886
14	1	Иванов	1887
15	9	Крокодил	1865
16	9	Вечный муж	1870
17	1	Трагик поневоле	1889
18	12	Воспоминания детства	1890
19	9	Записки из подполья	1864
20	2	Защита Лужина	1930
21	12	Воспоминания о Джордже Эллиоте	1886
22	2	Подвиг	1932

Получим информацию о книгах, опубликованных в 1866 году, и об авторах этих книг. Для этого составим запрос, который вернет нам из таблицы *author* имя автора, а из таблицы *book* — название и год издания литературного произведения.

```
SELECT
  -- Поля, которые мы хотим получить в ответе
  author.name,
  book.title,
  book.year
FROM
  -- Таблицы, из которых запрашиваем данные:
  author,
  book
WHERE
  -- В таблице book найти строки, в которых поле year содержит "1886"
  book.year = 1886
  AND
  -- И значение поля book.author_id должно быть равно значению
author.id
  book.author_id = author.id;
```

Результирующая выборка, полученная по нашему запросу:

```
name                title                                                year
----------------  ------------------------------------------------------
------
Лев Толстой         Власть тьмы, или Коготок увяз, всей птичке пропасть  1886
Антон Чехов         О вреде табака                                       1886
Софья Ковалевская   Воспоминания о Джордже Эллиоте                       1886
```

JOIN-запросы

Прекрасный способ сделать выборку из нескольких таблиц — это запрос с ключевым словом **JOIN.** В JOIN-запросе можно указать тип связи между таблицами и условие их объединения.

Тип связи для JOIN может быть указан ключевыми словами **FULL, OUTER/INNER, LEFT/RIGHT**. Если тип связи явно не указан, то применяется **INNER JOIN**.

В запросе блок **JOIN** идёт сразу после блока **FROM**:

```
SELECT

    -- Имена столбцов результата

    Таблица1.ИмяСтолбца1,

    ...

FROM

    -- Запрашиваем данные ИЗ таблицы Таблица1

    Таблица1

-- к результатам из Таблица1 ПРИСОЕДИНИТЬ данные из таблицы Таблица2

JOIN Таблица2

    -- Условия объединения

    ON Таблица1.ИмяСтолбцаN = Таблица2.ИмяСтолбцаM

WHERE

    -- Дополнительные условия
```

author

id	name	birth_year
1	Антон Чехов	1860
2	Владимир Набоков	1899
3	Лев Толстой	1828
4	Насон Грядущий	3019
5	Александр Грибоедов	1795
6	Александр Островский	1823
7	Марина Цветаева	1892
8	Александр Герцен	1812
9	Фёдор Достоевский	1821
10	Николай Некрасов	1821
11	Николай Чернышевский	1828
12	Софья Ковалевская	1850

book

id	author_id	title	year
1	5	Горе от ума	1824
2	8	Кто виноват?	1846
3	7	Волшебный фонарь	1912
4	6	Утро молодого человека	1850
5	6	Бедная невеста	1851
6	8	Былое и думы	1852
7	11	Что делать?	1863
8	3	Заражённое семейство	1864
9	3	Нигилист	1866
10	10	Кому на Руси жить хорошо	1877
11	7	Вечерний альбом	1910
12	3	Власть тьмы, или Коготок увяз, всей птичке пропасть	1886
13	1	О вреде табака	1886
14	1	Иванов	1887
15	9	Крокодил	1865
16	9	Вечный муж	1870
17	1	Трагик поневоле	1889
18	12	Воспоминания детства	1890
19	9	Записки из подполья	1864
20	2	Защита Лужина	1930
21	12	Воспоминания о Джордже Эллиоте	1886
22	2	Подвиг	1932

spouse

id	name	author_id	wed_year
1	Мария Бахметьева	6	1869
2	Фёкла Викторова	10	1877
3	Сергей Эфрон	7	1912
4	Ольга Васильева	11	1853
5	Мария Исаева	9	1857
6	Наталья Захарьина	8	1838
7	Вера Слоним	2	1925
8	Софья Берс	3	1862
9	Нино Чавчавадзе	5	1828
10	Ольга Книппер	1	1901
11	Владимир Ковалевский	12	1868
12	Анна Сниткина	9	1867

Так вы уже умеете, вы делали это в прошлом уроке:

```
SELECT
      author.name,
      book.title,
      book.year
    FROM
      author,
      book
    WHERE
      book.year = 1886 AND book.author_id = author.id;
```

Аналогичный запрос с ключевым словом JOIN выглядит так:

```
SELECT
-- ВЫБРАТЬ данные из колонок //указываем имена колонок вместе с именем таблицы//
```

```
    author.name,
    book.title,
    book.year
FROM
    -- Запрашиваем данные ИЗ ТАБЛИЦЫ author
    author
-- к результатам ПРИСОЕДИНИТЬ данные из таблицы book
JOIN book
    -- показать только те записи, в которых
    -- значение поля book.author_id равно значению поля author.id
    ON book.author_id = author.id
WHERE
    -- из всего найденного показать только те результаты,
    -- где значение поля book.year равно "1886"
    book.year = 1886;
```

Результат обоих запросов будет одинаков:

```
name                title                                                year
----------------  --------------------------------------------------   ----------
Лев Толстой         Власть тьмы, или Коготок увяз, всей птичке пропасть  1886
Антон Чехов         О вреде табака                                       1886
Софья Ковалевская   Воспоминания о Джордже Эллиоте                       1886
```

Но только JOIN может без особых ухищрений сделать запрос «SQL, найди мне всех авторов в таблице author, у которых нет ни одной книги в таблице book»: `sql SELECT author.id, author.name, author.birth_year FROM author LEFT JOIN book ON book.author_id = author.id WHERE book.title IS NULL;`

Результат:

Скопировать кодSQL

```
   id  name            birth_year

    --  --------------  ----------

    4   Насон Грядущий  3019
```

При обработке запроса JOIN в вычислениях участвуют две таблицы, которые условно называют **«левая»** и **«правая»**. «Левая» — это та, которая вызвана в блоке FROM, «правая» указывается после ключевого слова JOIN.

Запрос JOIN позволяет назначить одну из таблиц «главной», а из другой таблицы вывести данные, связанные с найденными в «главной».

Тип этой связи может быть указан ключевыми словами **FULL OUTER, INNER, LEFT, RIGHT**. По умолчанию применяется **INNER**.

Ключевое слово назначает «главной» таблицей «левую» (при **LEFT JOIN**) или «правую» (при **RIGHT JOIN**). В примере запроса о писателях без книг «левая» таблица — это `author`, а «правая» — `book`.

При запросах **FULL OUTER JOIN** и **INNER JOIN** таблицы равнозначны, «главной» нет.

Вот фрагмент базы данных детского сада. В *таблице А* собраны мальчики, в *таблице Б* — девочки. Дети одеты в карнавальные костюмы.

```
-- Табл. А    Табл. Б

id costume    id  costume

-- --------   --  ---------

1  Пират      1   Жучка

2  Снежинка   2   Пират

3  Котик      3   Принцесса
```

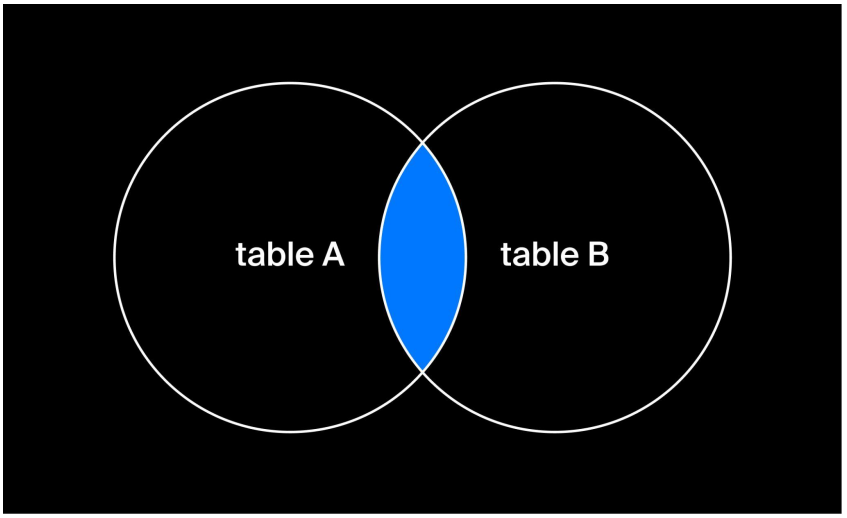
```
4   Буратино    4    Котик
```

Дети не очень понимают как себя вести, но воспитатель хочет, чтобы они все вместе начали что-то делать. В зависимости от задач дети будут разбиваться на пары или собираться в какие-то группы.

- **INNER JOIN** выберет из перечисленных таблиц только те записи, у которых совпадают значения заданных в условии **ON** полей. Подойдёт для танцев, где каждому участнику нужна пара в таком же костюме. Находим одинаковых и отправляем на сцену, остальных не показываем. Пусть сидят у стенки на стульчиках.

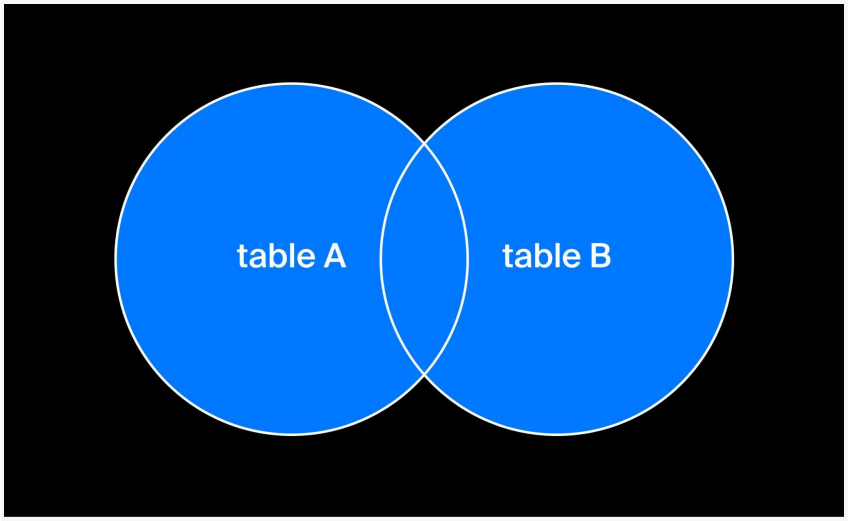
```
-- вернуть все поля из таблицы TableA

SELECT * FROM TableA

-- дополнительно запросить данные из таблицы TableB

INNER JOIN TableB

    -- и возвращать записи, в которых

    -- для значения TableA.costume найдено такое же в TableB.costume

    ON TableA.costume = TableB.costume;

-- Ответ:

id costume    id  costume

-- --------   --  ---------

1  Пират      2   Пират

3  Котик      4   Котик
```

Графическое представление **INNER JOIN**



- При запросе FULL OUTER JOIN выводятся все записи из обеих таблиц. Те записи, у которых запрошенные значения совпадают — выводятся парами, остальные выводятся поодиночке. Всё примерно так же, как в прошлом примере, но те, у кого нет пары — тоже выходят на сцену.

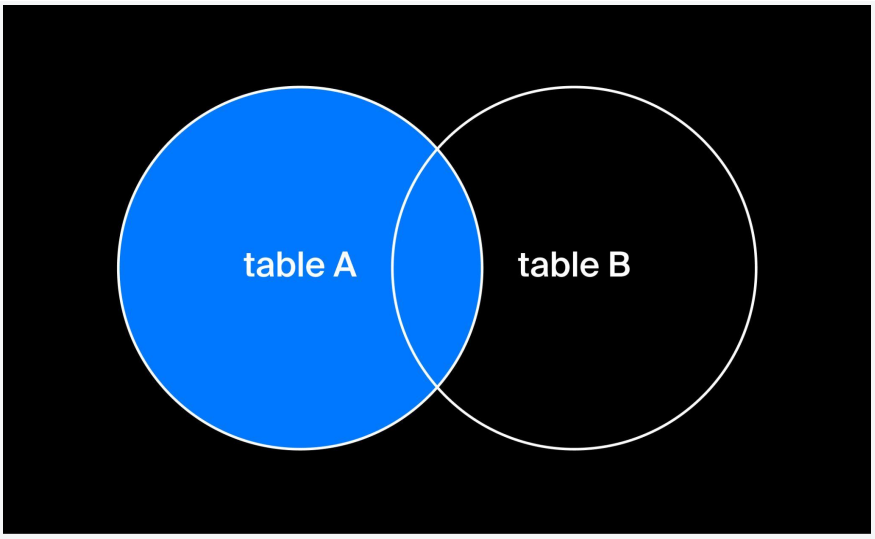
```
SELECT * FROM TableA
FULL OUTER JOIN
    TableB
ON
    TableA.costume = TableB.costume;

-- Ответ:

id    costume     id    costume
----  --------    ----  ---------
1     Пират       2     Пират
2     Снежинка    null  null
null  null        1     Жучка
3     Котик       4     Котик
null  null        3     Принцесса
4     Буратино    null  null
```



- **LEFT JOIN** — в результат попадают все подходящие записи из левой таблицы, а из правой — только записи, отвечающие условию в блоке **ON**. Всех мальчиков просят выйти и стать в ряд, и только те девочки, для которых есть пара, могут стать рядом с мальчиками.

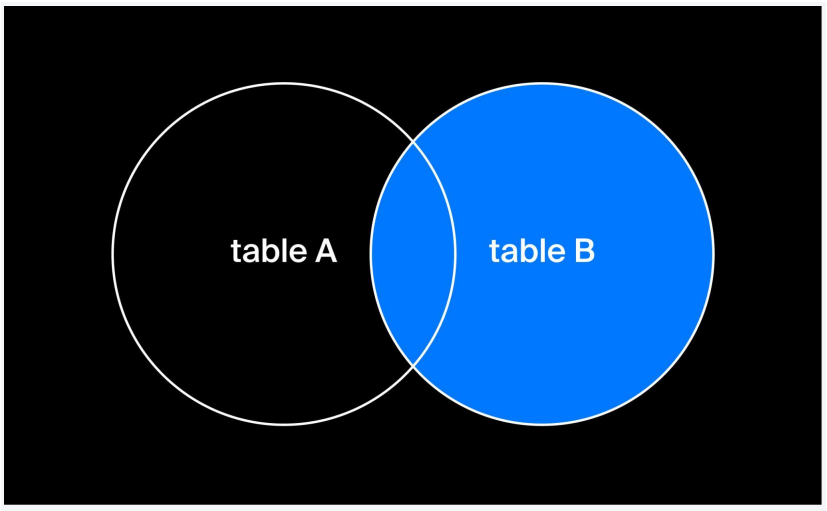
```
SELECT * FROM TableA
LEFT JOIN
    TableB
ON
    TableA.costume = TableB.costume;
```

```
-- Ответ:

id costume     id     costume
-- --------    ----   ---------
1  Пират       2      Пират
2  Снежинка    null   null
3  Котик       4      Котик
4  Буратино    null   null
```

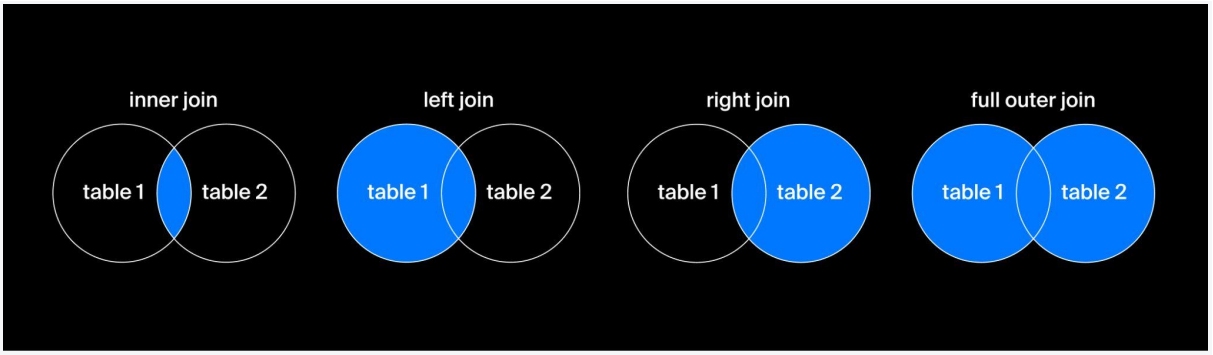


- **RIGHT JOIN** — в результат попадают все подходящие записи из правой таблицы, а из левой — только записи, отвечающие условию в блоке **ON**. Все девочки выходят на сцену, а мальчиков выпускают только тех, для кого есть пара.

```
SELECT * FROM TableA
RIGHT JOIN
    TableB
ON
    TableA.costume = TableB.costume;

-- Ответ:

id    costume     id     costume
----  --------    ----   ---------
null  null         1     Жучка
1     Пират        2     Пират
null  null         3     Принцесса
3     Котик        4     Котик
```



В зависимости от конкретной базы синтаксис JOIN-запросов может меняться, но обычно есть возможность выбрать необходимую комбинацию модификаторов LEFT, RIGHT, FULL и OUTER для создания необходимого JOIN-запроса.



Изменение типов

Некоторые агрегирующие функции работают только с числовыми типами данных. Потому такой запрос не выполнится для *books:*

```
SELECT
    AVG(pages) AS average
FROM
    Books;
```

Взгляните на таблицу:

books

ID	NAME	GENRE	AUTHOR	DATE_PUB	PAGES	PRICE
1	Капитанская дочка	5	Александр Пушкин	1836-01-01	130	150
2	Отцы и дети	1	Иван Тургенев	1862-01-01	240	207
3	Вишнёвый сад	7	Антон Чехов	1903-01-01	60	138
4	Война и мир	1	Лев Толстой	1869-01-01	1274	

ID	NAME	GENRE	AUTHOR	DATE_PUB	PAGES	PRICE
8	Зелёная миля	9	Стивен Кинг	1996-01-01	420	190
9	Унесённые ветром	1	Маргарет Митчел	1936-01-01	624	358
10	Три товарища	1	Эрих Мария Ремарк	1936-01-01	470	196
11	На Западном фронте без перемен	4	Эрих Мария Ремарк	1929-01-01	200	273

ID	NAME	GENRE	AUTHOR	DATE_PUB	PAGES	PRICE
12	Триумфальная арка	1	Эрих Мария Ремарк	1945-01-01	520	220
13	Чёрный обелиск	1	Эрих Мария Ремарк	1956-01-01	480	420
14	Ночь в Лиссабоне	1	Эрих Мария Ремарк	1961-01-01	250	187
			Эрих			

ID	NAME	GENRE	AUTHOR	DATE_PUB	PAGES	PRICE
14	Ночь в Лиссабоне	1	Эрих Мария Ремарк	1961-01-01	250	187
15	Жизнь взаймы	1	Эрих Мария Ремарк	1959-01-01	240	240
16	Станция на горизонте	1	Эрих Мария Ремарк	1927-01-01	210	
17	Евгений Онегин	1	Александр Пушкин	1832-01-01	736	330

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** (от англ. «преобразование»):

```
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
```

Осталось познакомиться с типами данных в SQL.

Числовые типы данных

integer — целочисленный тип, аналогичный типу *int* в Python. В SQL 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

'Имя' — 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. В формате *timestamp* чаще всего хранят события, происходящие несколько раз за день. Например, логи пользователей сайта.

date — дата.

Логический

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

Вернёмся к «нерешаемой» задаче. Найдем среднее число страниц двумя способами:

```
SELECT
    AVG(pages :: integer) AS average
FROM
    books;
```

```
428,7058824
```

В данных о молочной продукции тоже есть ошибка. Данные поля *weight* записаны как строки, хотя на самом деле это числа. По такому поводу стоит написать баг-репорт.

Группируем данные

ID	NAME	GENRE	AUTHOR	DATE_PUB	PAGES	PRICE
1	Капитанская дочка	5	Александр Пушкин	1836-01-01	130	150
2	Отцы и дети	1	Иван Тургенев	1862-01-01	240	207
3	Вишнёвый сад	7	Антон Чехов	1903-01-01	60	138
4	Война и мир	1	Лев Толстой	1869-01-01	1274	

ID	NAME	GENRE	AUTHOR	DATE_PUB	PAGES	PRICE
4	Война и мир	1	Лев Толстой	1869-01-01	1274	
5	Анна Каренина	1	Лев Толстой	1877-01-01	1000	126
6	Цветы для Элджернона	4	Дэниел Киз	1959-01-01	250	280
7	Похороните меня за плинтусом...	5	Павел Санаев	1994-01-01	184	198

ID	NAME	GENRE	AUTHOR	DATE_PUB	PAGES	PRICE
8	Зелёная миля	9	Стивен Кинг	1996-01-01	420	190
9	Унесённые ветром	1	Маргарет Митчел	1936-01-01	624	358
10	Три товарища	1	Эрих Мария Ремарк	1936-01-01	470	196
11	На Западном фронте без перемен	4	Эрих Мария Ремарк	1929-01-01	200	273

ID	NAME	GENRE	AUTHOR	DATE_PUB	PAGES	PRICE
12	Триумфальная арка	1	Эрих Мария Ремарк	1945-01-01	520	220
13	Чёрный обелиск	1	Эрих Мария Ремарк	1956-01-01	480	420
14	Ночь в Лиссабоне	1	Эрих Мария Ремарк	1961-01-01	250	187
			Эрих			

ID	NAME	GENRE	AUTHOR	DATE_PUB	PAGES	PRICE
14	Ночь в Лиссабоне	1	Мария Ремарк	1961-01-01	250	187
15	Жизнь взаймы	1	Эрих Мария Ремарк	1959-01-01	240	240
16	Станция на горизонте	1	Эрих Мария Ремарк	1927-01-01	210	
17	Евгений Онегин	1	Александр Пушкин	1832-01-01	736	330

Так подсчитали количество книг, написанных Ремарком:

```
SELECT
    COUNT(name) AS cnt
FROM
    books
WHERE
    author = 'Эрих Мария Ремарк';
```

```
7
```

Как найти число книг каждого автора? Не писать же несколько запросов! Хорошо бы автоматически перебирать авторов, и сразу же считать количество их произведений. Для этого есть группировка.

Команду **GROUP BY** (англ. «группировать по») применяют, когда данные нужно разделить на группы по значениям полей.

Пример формата запроса с агрегирующей функцией и группировкой:

```
SELECT
    поле_1,
    поле_2,
    ...,
    поле_n,
    АГРЕГИРУЮЩАЯ_ФУНКЦИЯ(поле) AS here_you_are
```

```
FROM
    таблица
WHERE -- если необходимо
    условие
GROUP BY
    поле_1,
    поле_2,
    ...,
    поле_n;
```

После команды **GROUP BY** перечисляют все поля из блока SELECT. Саму агрегирующую функцию включать в блок GROUP BY не нужно — с ней запрос не выполнится.

Вернемся к примеру. В нашем случае агрегирующая функция — COUNT(). Команда GROUP BY поможет найти количество строк в разрезе авторов:

```
SELECT
    author,
    COUNT(name) AS cnt
FROM
    books
GROUP BY
    author;
```

AUTHOR	CNT
Александр Пушкин	2
Антон Чехов	1
Дэниел Киз	1
Иван Тургенев	1
Лев Толстой	2
Маргарет Митчелл	1
Павел Санаев	1
Стивен Кинг	1

Теперь мы знаем, сколько книг написал каждый из авторов таблицы.

Подсчитаем количество книг в разрезе авторов и рейтинга:

```
SELECT
    author,
    rating,
```

```
    COUNT(name) AS cnt
FROM
    books
GROUP BY
    author,
    rating;
```

AUTHOR	RATING	CNT
Лев Толстой		1
Маргарет Митчелл		1
Павел Санаев	4.1	1
Эрих Мария Ремарк	4.1	1
Антон Чехов	4.4	1
Лев Толстой	4.6	1
Эрих Мария Ремарк	4.6	3
[illegible]	[illegible]	[illegible]

Эта конструкция вернула таблицу с количеством книг и их рейтингом определённого автора. Например, Эрих Мария Ремарк написал три книги с рейтингом 4.6, одну — с оценкой 4.1 и три с 4.7. А у Льва Толстого одно произведение с 4.6 баллами рейтинга и одно — вовсе без оценки.

Конструкция GROUP BY работает для всех агрегирующих функций: COUNT(), AVG(), SUM(), MAX(), MIN(). Можно вызывать несколько функций сразу. Например, для каждого автора найдем среднее количество страниц в его произведениях и максимальное количество страниц:

```
SELECT
    author,
    AVG(pages) AS avg_pages,
    MAX(pages) AS max_pages
FROM
    books
GROUP BY
    author;
```

AUTHOR	AVG_PAGES	MAX_PAGES
Александр Пушкин	433	736
Антон Чехов	60	60
Дэниел Киз	250	250
Иван Тургенев	240	240
Лев Толстой	1137	1274
Маргарет Митчелл	624	624
Павел Санаев	184	184
Стивен Кинг	489	489

Сортируем данные

Итоговые данные обычно представляют в определённом порядке. Чтобы сортировать данные по указанным полям, применяют команду **ORDER BY** (англ. *order by*, «упорядочить по»).

Формат запроса с группировкой и сортировкой:

```
SELECT
    поле_1,
    поле_2,
    ...,
    поле_n,
    АГРЕГИРУЮЩАЯ_ФУНКЦИЯ(поле) AS here_you_are
FROM
    таблица
WHERE -- если нужно
    условие
GROUP BY
    поле_1,
    поле_2,
    ...,
    поле_n,
ORDER BY -- если необходимо, перечисляем только те поля,
--по которым хотим отсортировать таблицу
    поле_1,
    поле_2,
    ...,
    поле_n,
    here_you_are;
```

В отличие от GROUP BY, в блоке с командой ORDER BY перечисляем только те поля, по которым хотим сортировать.

У команды ORDER BY два аргумента. Они отвечают за порядок сортировки в столбцах:

- **ASC** (от англ. *ascending*, «восходящий») сортирует данные в порядке возрастания. Это аргумент ORDER BY по умолчанию.
- **DESC** (от англ. *descending*, «нисходящий») сортирует данные по убыванию.

Аргументы команды ORDER BY указывают сразу после поля, по которому сортировали данные:

```
ORDER BY
    название поля DESC
-- сортируем данные по убыванию

ORDER BY
    название_поля ASC
-- сортируем данные по возрастанию
```

ID	NAME	GENRE	AUTHOR	DATE_PUB	PAGES	PRICE
1	Капитанская дочка	5	Александр Пушкин	1836-01-01	130	150
2	Отцы и дети	1	Иван Тургенев	1861-01-01	240	207
3	Вишнёвый сад	7	Антон Чехов	1903-01-01	60	138
4	Война и мир	1	Лев Толстой	1869-01-01	1274	
5	Анна	1	Лев	1878-01-	1000	126

Посчитаем количество книг каждого автора и отсортируем данные:

```
SELECT
    author,
    COUNT(name) AS cnt
FROM
    books
GROUP BY
    author
ORDER BY
    cnt;
```

AUTHOR	CNT
Антон Чехов	1
Дэниел Киз	1
Иван Тургенев	1
Маргарет Митчелл	1
Павел Санаев	1
Стивен Кинг	1
Александр Пушкин	2
Лев Толстой	2

Мы не указали аргумента для ORDER BY, и данные отсортированы по умолчанию — в порядке возрастания количества книг.

Добавим аргумент DESC:

```
SELECT
    author,
    COUNT(name) AS cnt
FROM
    books
GROUP BY
    author
ORDER BY
    cnt DESC;
```

AUTHOR	CNT
Эрих Мария Ремарк	7
Александр Пушкин	2
Лев Толстой	2
Антон Чехов	1
Дэниел Киз	1
Иван Тургенев	1
Маргарет Митчелл	1
Павел Санаев	1

Сразу видно, какого автора предпочитает составитель таблицы! Аргумент DESC вывел данные в порядке убывания.

Команда **LIMIT** (англ. «предел») ограничивает количество строк в выводе. Её всегда указывают последней в запросе:

```
SELECT
    поле_1,
    поле_2,
    ...,
    поле_n,
    АГРЕГИРУЮЩАЯ_ФУНКЦИЯ(поле) AS here_you_are
FROM
    таблица
WHERE -- если необходимо
    условие
GROUP BY
    поле_1,
    поле_2,
    ...,
    поле_n,
ORDER BY -- если необходимо, перечисляем только те поля,
--по которых хотим отсортировать таблицу
    поле_1,
    поле_2,
    ...,
    поле_n,
    here_you_are
LIMIT -- если необходимо
    n;
-- n-максимальное количество строк, которое вернёт такой запрос
```

После LIMIT указывают требуемое число строк — n. Удобнее строить рейтинг из ограниченного числа элементов.

Например, выведем топ-3 книг по количеству страниц:

```
SELECT
    name,
    pages
FROM
    books
ORDER BY
    pages DESC
LIMIT
    3;
```

NAME	PAGES
Война и мир	1274
Анна Каренина	1000
Евгений Онегин	736

Операторы и функции для работы с датами

Бизнес-процессы тесно увязаны со временем.

Например, чтобы оценить влияние погоды на покупательскую активность в интернет-магазине, важно знать, когда пользователь совершил покупку и какая погода была в тот момент. Нужно уметь соединять данные о времени из разных источников и группировать информацию по месяцу, дню или часу.

Две основные функции для работы со временем и датой — **EXTRACT** (англ. «извлекать») и **DATE_TRUNC** (от англ. *truncate date*, «усекать, отбрасывать дату»). Обе функции вызывают в блоке SELECT.

Шаблон функции EXTRACT:

```
SELECT
      EXTRACT(часть_даты FROM столбец) AS новый_столбец_с_датой
    FROM
        Таблица_со_всеми_датами;
```

Название функции определяет ее суть. EXTRACT *извлекает* из даты нужную часть: год, месяц, минуту. Что ещё можно получить вызовом EXTRACT:

- `century` — век;
- `day` — день;
- `doy` (от англ. *day of the year*) — день года: от 1 до 365/366;
- `isodow` (от англ. *day of the week* и ISO 8601, международного стандарта даты и времени) — день недели: понедельник — 1, воскресенье — 7.
- `hour` — час;
- `milliseconds` — миллисекунда;
- `minute` — минута;
- `second` — секунда;
- `month` — месяц;
- `quarter` — квартал;
- `week` — неделя в году;

- `year` — год.

Вызовем функцию EXTRACT и получим из поля *log_on* таблицы *user_activity* (англ. «активность пользователей») два столбца — с месяцем и днём входа определенного пользователя в личный кабинет:

ID_USER	LOG_ON	LOG_OFF
6	2019-03-01 23:34:55	2019-04-01 01:20:45
156	2019-07-03 17:59:21	2019-07-03 19:31:34
65	2019-03-25 14:30:46	2019-03-25 17:47:53

```
SELECT
    id_user,
    EXTRACT(MONTH FROM log_on) AS month_activity,
    EXTRACT(DAY FROM log_on) AS day_activity
FROM
    user_activity;
```

USER_ACTIVITY	MONTH_ACTIVITY	DAY_ACTIVITY
6	3	1
156	7	3
65	3	25

Пользователь под номером 6 входил в личный кабинет первого числа третьего месяца. А 156-ой пользователь — третьего числа седьмого месяца.

DATE_TRUNC *усекает* дату до часа, дня или месяца. В отличие от EXTRACT часть, до которой нужно усечь дату, записывают как строку. А столбец, откуда берут данные о времени, указывают через запятую:

```
SELECT
    DATE_TRUNC('часть_даты_до_которой_усекаем', столбец) AS
новый_столбец_с_датой
FROM
    Таблица_со_всеми_датами;
```

Часть даты, до которой данные нужно «обнулить», указывают в аргументе функции DATE_TRUNC:

- `'microseconds'` — микросекунды;
- `'milliseconds'` — миллисекунды;
- `'second'` — секунда;
- `'minute'` — минута;
- `'hour'` — час;
- `'day'` — день;
- `'week'` — неделя;
- `'month'` — месяц;
- `'quarter'` — квартал;
- `'year'` — год;
- `'decade'` — декада года;
- `'century'` — век.

Знать точное время активности пользователя бывает важно. Однако, если нужно сгруппировать данные по дате и часу входа, минуты и секунды портят картину. Сократим значения поля *log_on* до часов.

```
SELECT
    DATE_TRUNC('hour',log_on) as date_log_on
FROM
    user_activity;
```

```
date_log_on

2019-03-01 23:00:00

2019-07-03 17:00:00

2019-03-25 14:00:00
```

Обработка данных в группировке

Вы уже умеете группировать по нескольким полям конструкцией GROUP BY.

Что если нужно сразу вывести авторов, у которых в таблицу попало больше одной книги? Для таких целей есть конструкция **HAVING** (англ. «обладающий»).

Пример формата запроса с конструкцией HAVING:

```
SELECT
    поле_1,
    поле_2,
    ...,
    поле_n,
    АГРЕГИРУЮЩАЯ_ФУНКЦИЯ(поле) AS here_you_are
FROM
    TABLE
WHERE -- если необходимо
    условие
GROUP BY
    поле_1,
    поле_2,
    ...,
    поле_n
HAVING
    АГРЕГИРУЮЩАЯ_ФУНКЦИЯ(поле_для_группировки) > n
ORDER BY -- если необходимо, перечисляем только те поля,
--по которых хотим отсортировать таблицу
    поле_1,
    поле_2,
    ...,
    поле_n,
    here_you_are
LIMIT -- если необходимо
    n;
```

В результирующую выборку попадут только те строки, для которых результат агрегирующей функции соответствует условию блоков HAVING и WHERE.

Если команды HAVING и WHERE так похожи, почему нельзя записать все условия в одной из них? Дело в том, что WHERE отрабатывает перед группировкой данных и расчетом агрегирующей функции.

Потому задать фильтр на результат агрегирующей функции в WHERE нельзя. И здесь выручает HAVING.

Таблица с книгами:

ID	NAME	GENRE	AUTHOR	DATE_PUB	PAGES	PRICE
1	Капитанская дочка	5	Александр Пушкин	1836-01-01	130	150
2	Отцы и дети	1	Иван Тургенев	1861-01-01	240	207
3	Вишнёвый сад	7	Антон Чехов	1903-01-01	60	138
4	Война и мир	1	Лев Толстой	1869-01-01	1274	
5	Анна	1	Лев	1878-01-	1000	126

Вернемся к задаче с авторами. Выберем только тех, у кого больше одной книги:

```
SELECT
    author,
    COUNT(name) AS name_cnt
FROM
    books
GROUP BY
    author
HAVING
    COUNT(name) > 1
ORDER BY
    name_cnt DESC;
```

NAME	NAME_CNT
Эрих Мария Ремарк	7
Александр Пушкин	2
Лев Толстой	2

Обратите внимание на порядок команд:

1) GROUP BY;

2) HAVING;

3) ORDER BY.

Указывайте команды строго в этой последовательности, иначе запрос не выполнится.

Подзапросы

Подзапрос — это запрос в запросе. Например, ассистент по актерам собирает портфолио с фото и пробами артистов, подходящих на роль Гамлета в новом фильме. А затем режиссёр, изучив портфолио, приглашает на кастинг пятерых, с которыми готов работать. Деятельность ассистента по актерам — это подзапрос, или

внутренний запрос. А выбор лучшего Гамлета из пяти — **внешний запрос**.

Подзапросы могут выполняться в разных частях запроса.

Если подзапрос записать в блоке FROM, то SELECT выбирает данные из таблицы, полученной в результате работы подзапроса. Имя этой таблицы указывают во внутреннем запросе, к ее столбцам обращаются во внешнем. Подзапрос записывают в круглых скобках:

```
SELECT
ПОДЗАПРОС_1.название_столбца,
ПОДЗАПРОС_1.название_столбца_2
FROM -- Для лучшей читабельности кода, переносите подзапрос на новую строку
    -- отделяйте подзапросы отступами
    (SELECT
        название_столбца,
        название_столбца_2
    FROM
        название_таблицы
    WHERE
        название_столбца = значение) AS ПОДЗАПРОС_1;
-- не забывайте давать имя подзапросу в блоке FROM
```

ID	NAME	GENRE	AUTHOR	DATE_PUB	PAGES	PRICE
1	Капитанская дочка	5	Александр Пушкин	1836-01-01	130	150
2	Отцы и дети	1	Иван Тургенев	1861-01-01	240	207
3	Вишнёвый сад	7	Антон Чехов	1903-01-01	60	138
4	Война и мир	1	Лев Толстой	1869-01-01	1274	
5	Анна	1	Лев	1878-01-	1000	126

Напишем запрос, который подсчитает среднее количество проставленных оценок по жанрам книг. Нужно вызвать две агрегирующие функции: COUNT() найдёт количество, AVG() подсчитает среднее. Однако написать `SELECT AVG(COUNT(rating))` не

выйдет — получим сообщение об ошибке: «агрегирующие функции не могут быть вложенными» (англ. *ERROR: aggregate function calls cannot be nested*).

Сначала нужно вызвать актеров на пробы, а потом просмотреть пробы и выбрать лучшего. Найдем количество оценок по жанрам в подзапросе, а во внешнем запросе рассчитаем среднее полученных значений:

```
SELECT
    AVG(Sub.count_rating) AS avg_count_rating
FROM
    (SELECT
        COUNT(rating) AS count_rating
    FROM
        books
    GROUP BY genre) AS Sub;
```

Разберём результат работы внутреннего запроса:

COLUMN 1	COLUMN 2
5	2
1	9
7	1
4	2
9	1

В внутреннем запросе подсчитали число оценок книг каждого жанра и сохранили его в поле count_rating. К таблице-результату работы подзапроса теперь можно обратиться во внешнем запросе.

Вызовем столбец *count_rating* таблицы *Sub* (от англ. *subquery*, «подзапрос») и найдем среднее функцией AVG():

```
3
```

Что это значит? Таблица содержит в среднем по 3 оценки, проставленные каждому жанру.

Внутренние запросы могут понадобиться в разных блоках внешнего запроса. Например, устроим подзапрос в блоке WHERE. Тогда выберутся данные из столбца со значениями, сгенерированными в результате работы подзапроса:

```
SELECT
    название_столбца,
    название_столбца_1
FROM
    название_таблицы
WHERE
    название_столбца =
        (SELECT
            столбец_1
        FROM
            название_таблицы_2
        WHERE
            столбец_1  = значение);
```

Дополним шаблон конструкцией IN, чтобы собирать данные из нескольких столбцов:

```
SELECT
    название_столбца,
    название_столбца_1
FROM
    название_таблицы
WHERE
    название_столбца IN
            (SELECT
                столбец_1
            FROM
                название_таблицы_2
            WHERE
                столбец_1 = значение_1 OR столбец_1 = значение_2);
```

Пример: добавим к таблице *books* таблицу *genre* с кодами жанров и их названиями:

ID	NAME
1	Роман
2	Басня
3	Комедия
4	Рассказ
5	Повесть
6	Поэма
7	Пьеса
8	Научно-образовательная

Выберем книги жанра «Роман» из таблицы books:

```
SELECT
    name,
    genre
FROM
    books
WHERE
    genre =
            (SELECT
                id
            FROM
                genre
            WHERE
                name ='Роман');
```

COLUMN 1	COLUMN 2
1	Роман

Подзапрос указывает, что нужно выбрать из таблицы *genre* те *id*, которым соответствует жанр — «Роман». Результат подзапроса: 1. Лишь одно значение из таблицы *genre* означает романы.

Перейдем к внешнему запросу. Он выбирает названия и жанры из таблицы *books.* Но только те, чей жанр равен результату внутреннего запроса, то есть единице.

Результат работы внешнего запроса:

NAME	GENRE
Отцы и дети	1
Война и мир	1
Анна Каренина	1
Унесенные ветром	1
Три товарища	1
Триумфальная арка	1
Черный обелиск	1
[illegible]	1

Напишем похожий запрос. Выберем не только романы, но и повести с драмами. Вот где пригодится IN:

```
SELECT
    name,
    genre
FROM
    books
WHERE
    genre IN
            (SELECT
                id
            FROM
                genre
            WHERE
                name IN ( 'Роман', 'Драма', 'Повесть' ));
```

Конструкция IN во внутреннем запросе выбирает *id* из таблицы жанров с названиями `'Роман'`, `'Драма'` или `'Повесть'`. В результате получим три *id*: 1, 9, 5. Эти значения передаются во внешний запрос.

Внешний запрос сообщает, что нужно выбрать названия и жанры из таблицы *books*, где жанры равны значениям из результирующего списка внутреннего запроса. То есть, 1, 9 или 5.

NAME	GENRE
Капитанская дочка	5
Отцы и дети	1
Война и мир	1
Анна Каренина	1
Похороните меня за плинтусом...	5
Зеленая миля	9
Унесенные ветром	1
Три товарища	1