**MY SQL**

Чтобы управлять базой данных, применяют специальные инструменты — **СУБД**, системы управления базами данных. Это комплекс программ, который позволяет создать базу данных, наполнить её новыми таблицами, отобразить содержимое, редактировать существующие таблицы.

Список популярных СУБД:

* PostgreSQL
* Oracle
* MySQL
* Microsoft SQL Server

**SQL** (от англ. Structured Query Language, «язык структурированных запросов») — язык запросов, с помощью которого можно управлять данными в реляционной базе.

Вот некоторые особенности языка:

Комментарии — это текст, который вставляют в запрос, чтобы объяснить, что именно делает этот код.

Начало однострочного комментария обозначают двумя дефисами: --

-- однострочный комментарий на языке SQL

Многострочный комментарий заключают в /\* слеши со звёздочками \*/:

/\* многострочный

комментарий

называют

так,

потому что

в нём

много

строк \*/

Команды принято писать заглавными буквами — так удобнее читать:

SELECT, WHERE, FROM

Запрос — одна команда в базе данных. 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 ";"

Перенос строки принято делать 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;

**Запросы**

**Запрос** — это команда к базе данных, которая написана в синтаксисе SQL. В запросе пишут, например, какие данные выбрать, как именно их обработать или изменить.

Оператор — это специальное слово или символ в синтаксисе SQL для выполнения операции над данными.

**Оператор SELECT**

Выборку данных можно получить, если применить оператор SELECT (англ. «выбирать»). Синтаксис запроса с SELECT:

SELECT

    название\_столбца\_1,

    название\_столбца\_2,

    название\_столбца\_3 ...

FROM

    название\_таблицы;

-- *выбрать один столбец из таблицы*

Чтобы выбрать из таблицы все столбцы, укажи в операторе SELECT звёздочку \* вместо названий столбцов:

SELECT \*

FROM books;

# Срезы данных в SQL

Условие в SQL-запросе, которое покажет ограниченную выборку, прописывают командой **WHERE** (англ. «где»). Оператор проверяет, соответствует ли каждая строчка таблицы условию, и выбирает подходящие.

SELECT

    название\_столбца\_1,

    название\_столбца\_2 -- *выбери названия столбцов*

FROM

    название\_таблицы -- *укажи таблицу*

WHERE

    условие; -- *определи условие, по которому отбираешь строки*

Порядок операторов строго определён. Они должны идти так:

1) SELECT

2) FROM

3) WHERE

Обозначить границы выборки в условиях помогают математические операторы:

|  |  |
| --- | --- |
| НАЗВАНИЕ | СМЫСЛ |
| = | равно |
| <>, != | не равно |
| > | больше |
| < | меньше |
| >= | больше или равно |
| <= | меньше или равно |

В SQL есть логические операторы: AND, OR, NOT.

**Операторы**

|  |  |
| --- | --- |
| НАЗВАНИЕ | ОПИСАНИЕ |
| **AND** | позволяет выбрать строки, которые соответствуют всем условиям. |
| **OR** | позволяет выбрать строки, которые подходят любому условию в запросе. |
| **NOT** | позволяет выбрать строки, которые не соответствуют определённому условию. |

Теперь нужно ограничить выборку одновременно 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*

Теперь нужно написать код с оператором IN (англ. «в, внутри»). После IN указывают список значений, которые нужно включить в результат:

 SELECT

    \*

FROM

    название\_таблицы

WHERE

    название\_столбца IN (&apos;значение\_1&apos;,&apos;значение\_2&apos;,&apos;значение\_3&apos;);

Если в списке должны быть числа, их указывают через запятую: IN (3,7,9). Строки тоже через запятую, но в одинарных кавычках: IN (&apos;значение\_1&apos;,&apos;значение\_2&apos;,&apos;значение\_3&apos;). Дату и время обозначают так: IN (&apos;ГГГГ-мм-дд&apos;,&apos;ГГГГ-мм-дд&apos;)

# Агрегирующие функции

В SQL есть функции, которые помогают посчитать общее количество строк в результате запроса, сумму, среднее значение, максимум и минимум для определённого поля в ответе. Такие функции называются агрегирующими. Все записи таблицы фильтруются, а затем к выборке применяется агрегирующая функция.

Как может выглядеть запрос 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 того как ты вызовешь агрегирующую функцию, имя столбца выведется в неудобном виде. Чтобы исправить это, примени команду AS (англ. «как») и запиши подходящее имя столбца.

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

В зависимости от цели 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.

В результате вернётся число:

* всех строк таблицы;
* строк, в которых указан рейтинг;
* уникальных значений rating.

Функция SUM**(column)** возвращает сумму по столбцу column. Когда функция выполняется, пропуски игнорируются.

функция SUM() работает только с числовым форматом данных.

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

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

Как изменить тип данных столбца в 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.

### Числовые типы данных

**integer** — целочисленный тип

**real** — число с плавающей точкой. Точность числа типа real до 6 десятичных разрядов.

### Строковые типы данных

Пример: **&apos;Практикум&apos;** — значение строкового типа, в SQL-запросе его заключают в одинарные кавычки.

**varchar(n)** — строка переменной длины, где **n** — ограничение. Этот тип данных варьирует длину хранимой строки: в поле можно занести любую строку короче, чем **n** символов.

**text** — строка любой длины.

### Дата и время

Любые значения даты или времени заключают в одинарные кавычки.

**timestamp** — дата и время. В формате timestamp чаще всего хранят события, для которых недостаточно указать только дату. Например, в логах указывают дату и точное времяпроизошедшего события. Требуется точность до секунды.

Пример: &apos;1990-01-01 00:00:00&apos;

**date** — дата. Имеет точность до дня без поддержки часового пояса.

Пример: &apos;1990-01-01&apos;

### Логический

**boolean** — логический тип данных. В PostgreSQL есть три варианта значений:

**TRUE** — «истина»,

**FALSE** — «ложь»,

**NULL** — «неизвестно».

# Группировка данных

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

 Пример запроса с агрегирующей функцией и группировкой:

 SELECT

    поле\_1,

    поле\_2,

    ...,

    поле\_n,

    АГРЕГИРУЮЩАЯ\_ФУНКЦИЯ(поле) AS field\_name

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.

Агрегирующая функция следующего примера — COUNT().

# Сортируем данные

Чтобы сортировать данные по указанным полям, применяют команду ORDERBY (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 нужно перечислить только поля, по которым ты хочешь сортировать информацию.

У команды ORDER BY есть аргумент, регулирующий порядок сортировки в столбцах. Он может принимать такие значения:

* ASC (от англ. ascending, «восходящий») сортирует данные в порядке возрастания. Это значение аргумента 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.

Команда LIMIT (англ. «предел») ограничивает количество строк в выводе. Её указывают в последнюю очередь:

 SELECT

    поле\_1,

    поле\_2,

    ...,

    поле\_n,

    АГРЕГИРУЮЩАЯ\_ФУНКЦИЯ(поле) AS field\_name

FROM

    таблица

WHERE -- если необходимо

    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

LIMIT -- *если необходимо*

    n;

*-- n-максимальное количество строк, которое вернёт такой запрос*

После LIMIT указывают нужное число строк — n.

### Оператор INSERT

INSERT помогает добавить новую строку в таблицу (англ. «вводить»). Синтаксис запроса:

INSERT INTO

название\_таблицы

(название\_столбца\_1, название\_столбца\_2, название\_столбца\_3 ...

-- *блок с названиями столбцов необязательный*

VALUES

(значение\_1, значение\_2, значение\_3...);  
 В запросе два ключевых оператора: INSERT INTO и VALUES.

INSERT INTO указывает, в какую таблицу добавить данные.

 VALUES перечисляет данные, которые нужно добавить.

# Изменение данных: UPDATE

### Оператор UPDATE

Оператор UPDATE (англ. «обновить») меняет текущую информацию в таблице. Синтаксис запроса:

UPDATE

название\_таблицы

SET

 имя\_столбца = значение\_поля

WHERE

условие; *-- определяем условие, по которому будем отбирать строки*

  В запросе три ключевых оператора: UPDATE, SET и WHERE.

UPDATE указывает, в какой таблице изменить данные. SET указывает столбец, в котором необходимо поменять данные. WHERE выбирает строки, опционально.

# Изменение данных: DELETE

### Оператор DELETE

Оператор \*\*DELETE\*\* удаляет записи из таблицы (англ. «удалять»). Синтаксис запроса:

DELETE FROM

название\_таблицы

WHERE

условие; *-- определяем условие, по которому будем отбирать строки*

В запросе два ключевых оператора: DELETE FROM и WHERE.

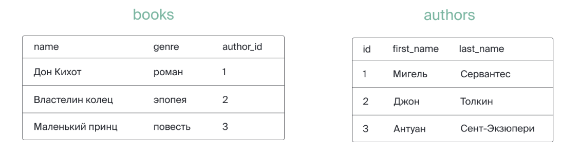
DELETE FROM указывает, из какой таблицы удалить записи.

 WHERE выбирает строки, в которых нужно удалить данные. Это опционально.

**Типы связей в таблицах**

Перед тобой две таблицы: books — с информацией о книгах и authors — с данными об авторах. Есть задача: нужно найти все книги, которые написал конкретный автор.

В таблице books вместо имени автора указан его id — так избегают дублирования информации. А в таблице authors есть имя автора и его фамилия.



Почему две таблицы?

Информацию о книгах хранят в одной таблице, а об авторах — в другой, потому что это позволяет избежать дублирования информации. В одной таблице около каждого произведения надо указать имя автора — это занимает память. Если таблица книг огромная, то потери памяти — тоже огромные. Поэтому информацию делят на две таблицы и указывают author\_id.

Как увидеть книгу и автора этой книги одновременно?

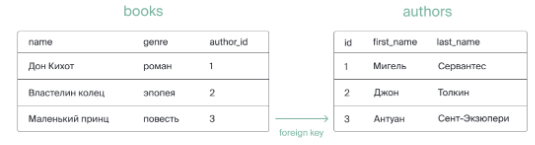
Ты можешь написать два запроса: выбрать id автора из таблицы authors по его имени и выбрать книги из таблицы books с нужным id. Но есть способ сделать это в одном запросе — связать таблицы.

Поле author\_id помогает точно определить, какой автор написал книгу: оно ссылается на id автора в таблице authors.



Когда столбец таблицы ссылается на поле с уникальными значениями другой таблицы, это называется **внешний ключ** (англ. foreign key).

Он отвечает за связь между таблицами. В этом примере внешний ключ — поле author\_id.

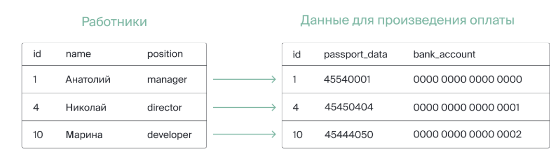


Связи между таблицами делят на три типа:

* «один к одному»;
* «один ко многим»;
* «многие ко многим».

**Один к одному** значит, что строка в первой таблице связана с одной-единственной строкой во второй таблице.

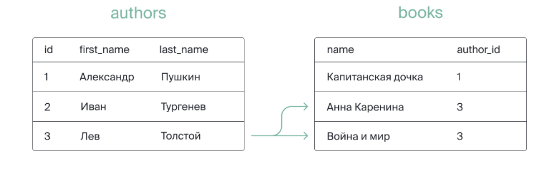
Например, перед тобой — таблица работников с именами и должностью, а также таблица с данными для зарплаты. Каждому сотруднику соответствует одна строка с номером паспорта и банковским счётом, на который её начисляют. Id сотрудников одинаковый в обеих таблицах.



Этот тип связи встречается редко: его применяют, чтобы защитить данные. Например, к списку сотрудников могут получить доступ все, а к данным на оплату — только бухгалтер. Одн ко многим

**Один ко многим** — тип связи, когда каждая строка в одной таблице соответствует многим строкам в другой таблице.

Такой тип связи есть в примере с книгами. Автор может написать несколько книг, но у одной книги — только один автор.



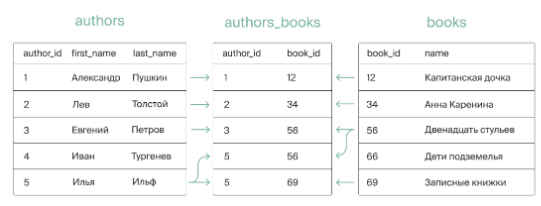
Бывает, что книги пишут несколько авторов. Например, Илья Ильф и Евгений Петров. Тогда нужен тип связи «многие ко многим»ко

**Многие ко многим** — тип связи, когда несколько строк одной таблицы соответствуют нескольким строкам другой таблицы. *\*\**

Например:

* есть таблица authors с авторами и первичным ключом author\_id;
* есть таблица books с информацией о книгах и первичным ключом book\_id.

Тебе надо найти книги соответствующих авторов. Чтобы это было возможно, нужна ещё одна таблица authors\_books: с её помощью сопоставляются таблицы authors и books.



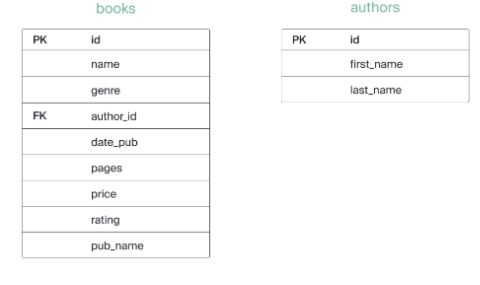
 Связь «многие ко многим» реализуется за счёт **связующей таблицы.** Она соединяет первичные ключи обеих таблиц. *\*\**

**ER-диаграммы**

Структуру базы данных иллюстрируют **ER-диаграммы** (от англ. entity-relationship diagram, диаграмма «объект — связь»). Они показывают, как таблицы связаны друг с другом.

**Таблицы**

Перед тобой — фрагмент ER-диаграммы:



Таблицы на ER-диаграммах переносят в виде двух колонок. В верхней части указывают название таблицы. В примере это books и authors.

В нижней части перечисляют поля таблицы. Также указывают, к каким ключам они относятся — первичным или внешним. Сами ключи обычно обозначают подписями **PK** (от англ. primary key, «первичный ключ») и **FK** (от англ. foreign key, «внешний ключ»). Часто вместо подписей ставят пиктограмму ключа или символ #.

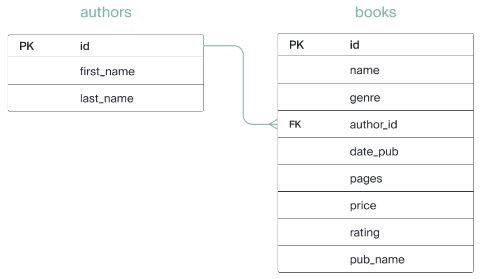
 Как прочитать первую таблицу ER-диаграммы?

* Таблица books содержит поля: id, name, genre, author\_id, date\_pub, pages, price, rating, pub\_name.
* Первичный ключ — поле id (PK).
* Внешний ключ — поле author\_id (FK).

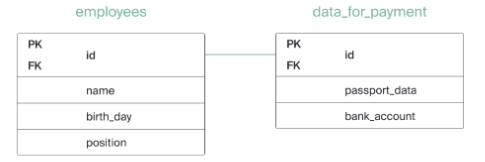
**Связи на ER-диаграммах**

Связи также отображают на ER-диаграммах. Место, соединяющее линию и таблицу, показывает, сколько значений одной таблицы соответствует одному или нескольким значениям другой.

Перед тобой первая таблица: тип связи — «один ко многим». Один автор может написать несколько книг, но здесь у книги — только один автор.

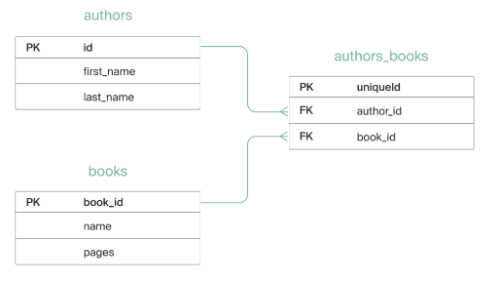


Вторая таблица. У одного сотрудника один набор реквизитов для зарплаты. То есть в таблицах будет один и тот же id — он одновременно выступает в роли внешнего и первичного ключа. Тип связи — «один к одному»:



Третья таблица. У книги может быть несколько авторов, и каждый мог написать несколько книг. То есть это две связи «один ко многим» с промежуточной таблицей authors\_books.

Тип связи — «многие ко многим»:



# Поиск пропусков в данных

Иногда в данных встречаются пропуски — нет нужной информации об объекте. В SQL пустые ячейки называют **NULL**. Чтобы их найти, применяют конструкцию **IS NULL** (от англ. is null, «является ничем»):

SELECT

     \*

FROM

     название\_таблицы

WHERE

    название\_столбца IS NULL;

IS имеет значение. Такая проверка на NULL вернёт не ту выборку, которую нужна:

SELECT

\*

FROM

     название\_таблицы

WHERE     название\_столбца = NULL;

Дело в том, что NULL показывает отсутствие значения, но сам значением не является. Конструкция IS NULL показывает, что значение отсутствует, но не сообщает о том, какое оно может быть.

# Поиск данных в таблице

Частая задача — выгрузить стоимость продукции определённого бренда. Что делать, если бренд указан в столбце с названием продукта, а отдельного поля нет? Надо уметь искать подстроки — ты уже знаешь, что это такое: тебе приходилось искать подходящие логи.

Оператор **LIKE** (англ. «подобный») находит схожие значения в таблице. Искать можно не только целое слово, но и его часть.

Синтаксис LIKE:

 название\_столбца LIKE ‘паттерн’

Перед оператором LIKE указывают столбец, в котором нужно искать, а после LIKE — шаблон для поиска — паттерн.

**Паттерн** (маска) — это шаблон, который позволяет найти целую строку по подстроке. Паттерны построены из символов, которые помогают заменить

 значение.

Например, знак нижнего подчёркивания \_ в паттерне заменяет одно подстановочное значение, то есть один символ. Знак процента % заменяет любое количество символов.

Тебе понадобится паттерн &apos;%Толст\_\_&apos;. Знак процента % заменит несколько символов в фамилии автора, а два знака нижнего подчёркивания \_ — окончание.

Паттерн.

Символы % и \_ позволяют искать подстроки более гибко. Но что, если эти символы есть в строках, которые ты ищешь?

Нужно рассказать системе, что это не «заменители» символов, а сами символы — % и \_. Когда ты сообщаешь системе «воспринимай % или \_ как символ, а не как заменитель», ты **экранируешь** их.

В SQL есть свой способ экранирования. Сначала выбирают экранирующий символ — его можно выбрать произвольно. Главное, чтобы он не встречался в таблице в сочетании с % или \_. Пусть это будет /. Следующий шаг — сказать системе, что слеш — теперь экранирующий символ. Это можно сделать оператором **ESCAPE** (от англ. «экранировать»):

WHERE units LIKE &apos;/%&apos; ESCAPE &apos;/&apos;

Если ты не укажешь экранирующий символ, выберутся все строки из таблицы, ведь % обозначает любое количество любых символов. Сейчас тебе удалось выделить только те строки, которые состоят из символа процента.

Вот пример, как найти все строки, заканчивающиеся на %. В качестве экранирующего символа выбран !:

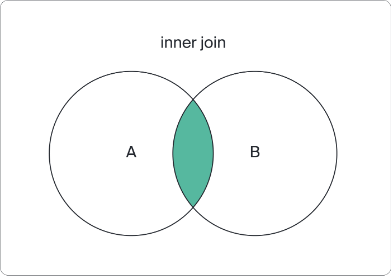
WHERE название\_столбца LIKE &apos;%!%&apos; ESCAPE &apos;!&apos;

*-- найдёт все строки, заканчивающиеся на %*

# JOIN. INNER JOIN.

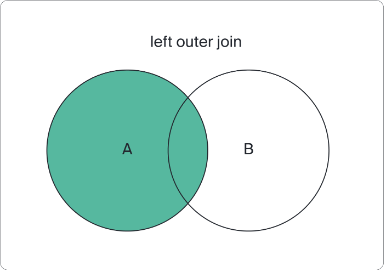
Не всегда удобно идти в одну таблицу, смотреть первичные ключи и искать их же во второй таблице. Удобнее получить понятный результат с помощью одного запроса. Чтобы это сделать, применяют соединение таблиц — используют оператор **JOIN** (англ. «соединять»). Таблицы можно соединить через **INNER** (англ. «внутренний») и **OUTER** (англ. «внешний»):

Соединение INNER возвращает строки строго на пересечении двух таблиц.

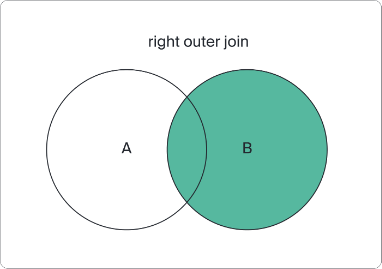


Различают три вида OUTER JOIN: LEFT, RIGHT и FULL.

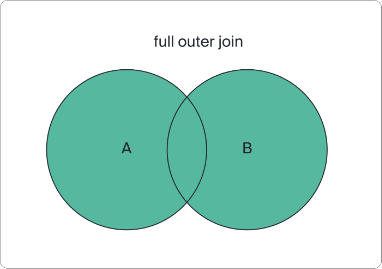
**LEFT OUTER JOIN**: возвращаются все строки левой таблицы. Если у строк левой таблицы выполняются условия соединения, они дополнятся данными правой таблицы. Если есть недостающие данные, вместо строк правой таблицы подставляются NULL-значения.



**RIGHT OUTER JOIN**: возвращаются все строки правой таблицы. Если у строк правой таблицы выполняется условие соединения, они дополнятся данными левой таблицы. Если есть недостающие данные, вместо строк левой таблицы подставляются NULL-значения.



**FULL OUTER JOIN**: возвращаются строки и левой, и правой таблиц. Если у строк левой таблицы и правой таблицы выполняется условие соединения, они объединятся в одну строку. Если есть строки, у которых не выполняется условие соединения, они дополнятся NULL-значениями.



**INNER JOIN**

INNER JOIN формирует выборку только из тех данных, у которых выполнено условие присоединения. От порядка присоединения таблиц результат не изменится.

Пример формата запроса с INNER JOIN:

SELECT *-- перечисляют только те поля, которые нужны*

   TABLE\_1.поле\_1 AS поле\_1,

TABLE\_1.поле\_2 AS поле\_2,

    ...

    TABLE\_2.поле\_n AS поле\_n

FROM

TABLE\_1

INNER JOIN TABLE\_2 ON TABLE\_2.поле\_1 = TABLE\_1.поле\_2;

 Синтаксис:

* INNER JOIN — название способа соединения, после него указывают имя таблицы, с которой нужно соединить таблицу из блока FROM;
* ON открывает условие присоединения: TABLE\_2.поле\_1 = TABLE\_1.поле\_2. Соединяют только те строки, которые соответствуют условию. В этом случае значения в поле\_1 и поле\_2 равны.

Так как поля в разных таблицах могут быть названы одинаково, к ним обращаются с указанием имени таблицы. Сначала название таблицы, а потом имя её поля: TABLE\_1.поле\_1.

# Внешнее объединение таблиц. LEFT JOIN

OUTER JOIN, или внешнее соединение таблиц, выполняют тремя способами:

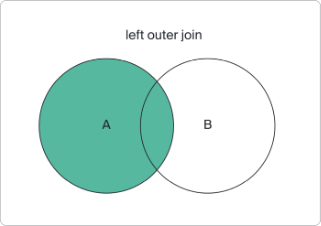
* **LEFT OUTER JOIN** (англ. «внешнее соединение налево»);
* **RIGHT OUTER JOIN** (англ. «внешнее соединение направо»);
* **FULL OUTER JOIN** (англ. «полное внешнее соединение»).

OUTER — опциональный оператор, который можно не указывать. Поэтому можно называть команды **LEFT JOIN,** **RIGHT JOIN, FULL JOIN**.

LEFT JOIN работает так: выполнится INNER JOIN — все строки левой и правой таблиц, которые могут быть связаны, свяжутся. После этого в выборку добавятся оставшиеся строки левой таблицы. Для них поля правой таблицы будут заполнены значением NULL.

RIGHT JOIN работает зеркально: выполнится INNER JOIN — все строки левой и правой таблиц, которые могут быть связаны, свяжутся. После этого в выборку добавятся оставшиеся строки правой таблицы. Для них поля левой таблицы будут заполнены значением NULL.

При FULL JOIN выберутся сначала пересекающиеся строки, затем строки левой таблицы, которые дополнятся NULL справа, а затем строки правой таблицы, которые дополнятся NULL слева.

Пример формата запроса с LEFT JOIN:

 SELECT     T

ABLE\_1.поле\_1 AS поле\_1,

    TABLE\_1.поле\_2 AS поле\_2,

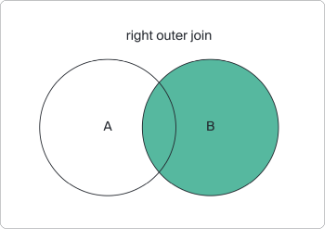
    ...

    TABLE\_2.поле\_n AS поле\_nFROM    TABLE\_1LEFT JOIN TABLE\_2 ON TABLE\_2.поле = TABLE\_1.поле;

 Как и в запросах с INNER JOIN, название таблицы указывают для каждого поля.

# Внешнее объединение таблиц. RIGHT JOIN

RIGHT JOIN аналогичен LEFT JOIN. В результат команда берёт всю *правую* таблицу и строки на пересечении с левой, если они подходят условию.



Пример формата запроса с RIGHT JOIN:

SELECT

    TABLE\_1.поле\_1 AS поле\_1,

    TABLE\_1.поле\_2 AS поле\_2,

    ...

    TABLE\_2.поле\_n AS поле\_n

FROM

    TABLE\_1

RIGHT JOIN TABLE\_2 ON TABLE\_1.поле = TABLE\_2.поле;

# Объединение нескольких таблиц

Пример формата запроса с несколькими INNER JOIN:

SELECT *-- перечисляем только те поля, которые нужны*

    TABLE\_1.поле\_1 AS поле\_1,

    TABLE\_1.поле\_2 AS поле\_2,

    ...

    TABLE\_3.поле\_n AS поле\_n

FROM

    TABLE\_1

INNER JOIN TABLE\_2 ON TABLE\_2.поле = TABLE\_1.поле

INNER JOIN TABLE\_3 ON TABLE\_3.поле = TABLE\_1.поле;

 К первой таблице присоединяют вторую, а потом третью. То есть все таблицы присоединяются к первой.

Соединять в одном запросе несколько таблиц можно не только методом INNER JOIN, но и RIGHT JOIN или LEFT JOIN.

**Практическая работа**

1. Установить MySQL Workbench.
2. Подключаемся через командную строку к mysql.
3. Посмотрим какие БД имеются.

**mysql> show databases;**

1. Выведен список БД.

+--------------------------+

| Database |

+--------------------------+

| information\_schema |

| mysql |

| performance\_schema |

| sakila |

| sys |

| test |

| world |

+--------------------------+

1. Выберем БД world

**mysql> use world;**

1. Посмотрим таблицы, содержащиеся в БД world.

**mysql> show tables from world;**

+---------------------+

| Tables\_in\_world |

+---------------------+

| city |

| country |

| countrylanguage |

+---------------------+

1. Посмотрим колонки в таблице city.

**mysql> SHOW COLUMNS FROM city;**

+----------------+------------+------+-------+---------+---------------------+

| Field | Type | Null | Key | Default | Extra |

+----------------+------------+------+-------+---------+---------------------+

| ID | int | NO | PRI | NULL | auto\_increment |

| Name | char(35) | NO | | | |

| CountryCode | char(3) | NO | MUL | | |

| District | char(20) | NO | | | |

| Population | int | NO | | 0 | |

+----------------+------------+------+-------+-----------+--------------------+

1. Выберем все данные из таблицы city.

**mysql> SELECT \***

**FROM city;**

+------+---------------------------+------------- +-------------- +-------------+

| ID | Name | CountryCode | District | Population |

+------+---------------------------+-------------+-------------------+-------------+

| 1 | Kabul | AFG | Kabol | 1780000 |

| 2 | Qandahar | AFG | Qandahar | 237500 |

| 3 | Herat | AFG | Herat | 186800 |

| 4 | Mazar-e-Sharif | AFG | Balkh | 127800 |

| 5 | Amsterdam | NLD | Noord-Holland | 731200 |

| 6 | Rotterdam | NLD | Zuid-Holland | 593321 |

……………………………………………………………………………

| 4073 | Gweru | ZWE | Midlands | 128037 |

| 4074 | Gaza | PSE | Gaza | 353632 |

| 4075 | Khan Yunis | PSE | Khan Yunis | 123175 |

| 4076 | Hebron | PSE | Hebron | 119401 |

| 4077 | Jabaliya | PSE | North Gaza | 113901 |

| 4078 | Nablus | PSE | Nablus | 100231 |

| 4079 | Rafah | PSE | Rafah | 92020 |

+------+--------------------------+--------------+-------------------+--------------+

1. Выведем данные для России

**mysql> SELECT \***

**FROM city**

**WHERE CountryCode = 'RUS';**

+------+---------------------------+---------- +----------------------+------------+

| ID | Name | CountryCode | District | Population |

+------+---------------------------+---------+------------------------+-------------+

| 3580 | Moscow | RUS | Moscow (City) | 8389200 |

| 3581 | St Petersburg | RUS | Pietari | 4694000 |

| 3582 | Novosibirsk | RUS | Novosibirsk | 1398800 |

| 3583 | Nizni Novgorod | RUS | Nizni Novgorod | 1357000 |

| 3584 | Jekaterinburg | RUS | Sverdlovsk | 1266300 |

| 3585 | Samara | RUS | Samara | 1156100 |

| 3586 | Omsk | RUS | Omsk | 1148900 |

| 3587 | Kazan | RUS | Tatarstan | 1101000 |

| 3588 | Ufa | RUS | Ba??kortostan | 1091200 |

| 3589 | T??eljabinsk | RUS | T??eljabinsk | 1083200 |

| 3590 | Rostov-na-Donu | RUS | Rostov-na-Donu | 1012700 |

| 3591 | Perm | RUS | Perm | 1009700 |

| 3592 | Volgograd | RUS | Volgograd | 993400 |

| 3593 | Voronez | RUS | Voronez | 907700 |

| 3594 | Krasnojarsk | RUS | Krasnojarsk | 875500 |

| 3595 | Saratov | RUS | Saratov | 874000 |

| 3596 | Toljatti | RUS | Samara | 722900 |

………………………………………………………………………………………

| 3764 | Kirovo-T??epetsk | RUS | Kirov | 91600 |

| 3765 | Krasnogorsk | RUS | Moskova | 91000 |

| 3766 | Klin | RUS | Moskova | 90000 |

| 3767 | T??aikovski | RUS | Perm | 90000 |

| 3768 | Novyi Urengoi | RUS | Yamalin Nenetsia | 89800 |

1. Вывести информацию по России, для тех городов, популярность которых , находится между 900 000 и 9 000 000.

**mysql> SELECT \***

**FROM city**

**WHERE CountryCode = 'RUS' AND Population**

**BETWEEN 900000 AND 9000000;**

1. Посчитать кол-во городов в каждой стране, вывести на экран коды стран с количеством городов > 40 и сгруппировать в убывающем порядке.

**mysql> SELECT CountryCode, COUNT(\*)**

**FROM city**

**GROUP BY CountryCode**

**HAVING COUNT(\*)>40**

**ORDER BY COUNT(\*) DESC;**

+----------------+--------------+

| CountryCode | COUNT(\*) |

+----------------+--------------+

| CHN | 363 |

| IND | 341 |

| USA | 274 |

| BRA | 250 |

| JPN | 248 |

| RUS | 189 |

| MEX | 173 |

| PHL | 136 |

| DEU | 93 |

| IDN | 85 |

| GBR | 81 |

| KOR | 70 |

| IRN | 67 |

| NGA | 64 |

| TUR | 62 |

| ESP | 59 |

| PAK | 59 |

| ITA | 58 |

| ARG | 57 |

| UKR | 57 |

| CAN | 49 |

| POL | 44 |

| ZAF | 44 |

| TWN | 42 |

| VEN | 41 |

+------------------+-------------+

1. Вывести первые пять строк из таблицы countrylanguage

**mysql> SELECT \***

**FROM countrylanguage**

**LIMIT 0, 5;**

1. Вывести из таблиц country и countrylanguage название страны, континент, регион, язык. Вывод ограничить первыми 5 строками. Объединение выполнить через WHERE

**mysql> SELECT country.name, country.Continent, country.Region, countrylanguage.Language**

**FROM country, countrylanguage**

**WHERE country.Code = countrylanguage.CountryCode**

**LIMIT 0, 5;**

+---------------+------------------+-------------------------------+--------------+

| name | Continent | Region | Language |

+---------------+------------------+--------------------------------+-------------+

| Aruba | North America | Caribbean | Dutch |

| Aruba | North America | Caribbean | English |

| Aruba | North America | Caribbean | Papiamento |

| Aruba | North America | Caribbean | Spanish |

| Afghanistan | Asia | Southern and Central Asia | Balochi |

+---------------+-------------------+--------------------------------+---------------+

1. Получить те- же данные, путем объединения таблиц через JOIN

**mysql> SELECT country.name, country.Continent, country.Region, countrylanguage.Language**

**FROM country**

**JOIN countrylanguage on country.Code = countrylanguage.CountryCode**

**LIMIT 0, 5;**