

# Конспект по теме "Отношения между таблицами"

## Типы связей в таблицах

Когда столбец таблицы содержит в себе значения поля другой таблицы, его называют **внешний ключ**. Он отвечает за связь между таблицами.

Существуют связи трёх типов:

- «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. По сути такая связь — расщепление одной таблицы на две. Это редкий тип связи, применяется в основном из соображений безопасности.

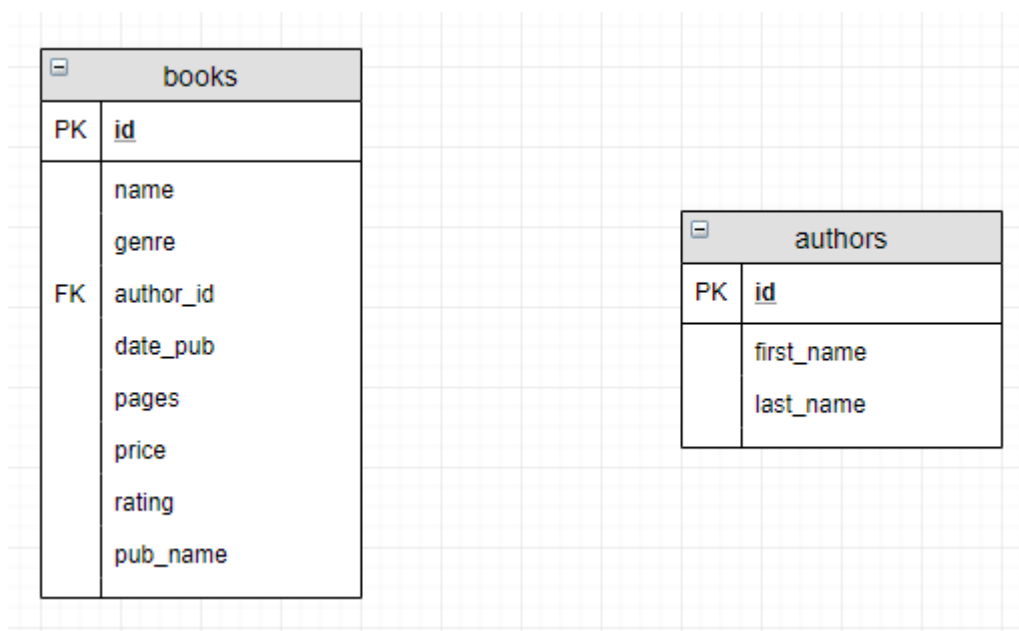
**Один ко многим** — тип связи, когда каждая строка в одной таблице соответствует многим строкам в другой таблице.

**Многие ко многим** — тип связи, когда несколько строк одной таблицы соответствуют нескольким строкам другой таблицы. Такая связь реализуется за счёт **стыковой таблицы**. Она соединяет первичные ключи обеих таблиц.

## ER-диаграммы

Устройство базы данных иллюстрируют **ER-диаграммы**. На них изображены таблицы и связи между ними.

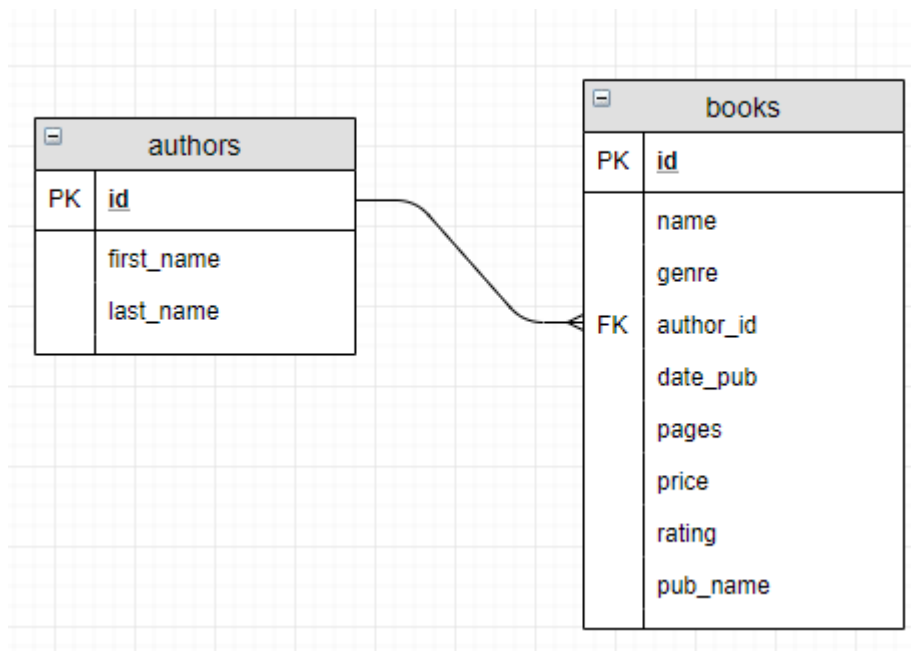
Таблицы на ER-диаграммах изображают прямоугольником, разделённым на два. В верхней части указывают название таблицы.



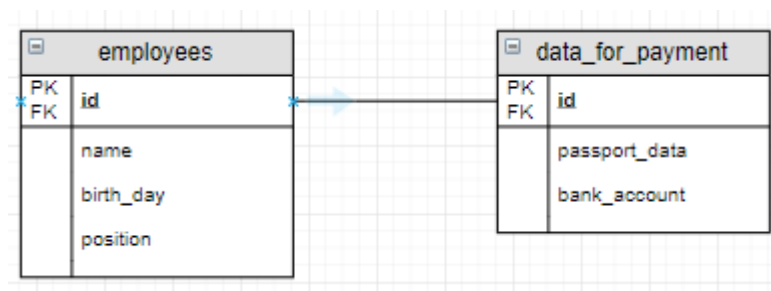
В нижней части перечисляют поля таблицы с указанием, к каким ключам они относятся — первичным или внешним. Сами ключи обычно обозначают подписями **PK** (первичный ключ) и **FK** (внешний ключ) . Иногда вместо подписей ставят пиктограмму ключа, символ решётки # или другой — по договорённости.

Связи тоже отображают на ER-диаграммах. Конец связующей линии показывает, одно или несколько значений одной таблицы соответствует значениям другой таблицы.

Пример связи «один ко многим»



Так изображают связь «один к одному»:



## Типовые роли пользователей баз данных

Продуктивную, или основную базу данных компании одновременно используют разные сотрудники. Каждому нужна только часть данных для решения собственной задачи. Чтобы многочисленные пользователи не мешали друг другу, их совместную работу с базой данных нужно организовать. Этим занимаются **администраторы баз данных**. Такие специалисты управляют доступом пользователей, отслеживают нагрузку, обеспечивают безопасность и создают резервные копии.

База данных — живой организм. Она постоянно растёт. «Здоровое» развитие базы обеспечивают **архитекторы** и **разработчики баз данных**. От результатов работы этих специалистов зависит структура данных, их

целостность и полнота. А также возможность её масштабировать: добавлять новые таблицы, связи и функции. Архитекторы и разработчики отвечают и за производительность базы данных.

Данные в базу добавляет **дата-инженер**. Это специалист по **ETL-процессам**: выгрузке, трансформации и пополнению данных базы.

Аналитик в этой схеме — простой пользователь. Он пишет запросы к базе и получает нужные данные, которые затем анализирует и применяет для проверки гипотез. Однако аналитик теснее всех работает с данными, а значит, первым сталкивается с нехваткой полей или таблиц. Обязательно сообщайте о таких открытиях разработчику, чтобы решить проблему быстро.

## Поиск пропусков в данных

SQL пустые ячейки называют **NULL**. Их ищут конструкцией **IS NULL**:

```
SELECT
  *
FROM
  название_таблицы
WHERE
  название_столбца IS NULL;
```

IS имеет значение. Такая проверка на NULL не пройдёт:

```
SELECT
  *
FROM
  название_таблицы
WHERE
  название_столбца = NULL; -- этот код не сработает!
```

Чтобы исключить из рассмотрения строки с NULL, применяют оператор **NOT**:

```
SELECT
  *
FROM
  название_таблицы
WHERE
  название_столбца IS NOT NULL;
```

Чтобы произвести действия в зависимости от некоторых условий, применяют конструкцию **CASE**. Она похожа на if-elif-else в Python:

```
CASE
  WHEN условие_1 THEN результат_1
  WHEN условие_2 THEN результат_2
  WHEN условие_3 THEN результат_3
  ELSE результат_4
END;
```

После оператора **WHEN** идёт условие. Если строка таблицы подходит условию, то возвращается результат, указанный в **THEN** (англ. «тогда»). Если нет, строка проходит проверку по следующему условию. В конце концов, если строка не соответствует ни одному из условий, описанных в **WHEN**, возвращается значение после **ELSE**, и конструкция **CASE** закрывается оператором **END**.

## Поиск данных в таблице

Оператор **LIKE** находит схожие значения в таблице. Искать можно не только целое слово, но и его часть.

Синтаксис **LIKE**:

```
название_столбца LIKE 'регулярное выражение'
```

Перед оператором **LIKE** указывают столбец, в котором нужно искать, а после **LIKE** —регулярное выражение.

Регулярные выражения в SQL несколько отличаются от регулярных выражений в Python. Например, знак нижнего подчёркивания `_` в регулярном выражении заменяет одно подстановочное значение, то есть 1 символ. Знак процента `%` заменяет любое количество символов. Внутри квадратных скобок `[]` указывается диапазон или последовательность символов, которые должны содержаться в строке. Если же необходимо, чтобы символы из диапазона или последовательности отсутствовали, используется конструкция `[^]`.

Диапазон или  
последовательность символов

Если нужно найти символ из регулярного выражения как подстроку, используется оператор **ESCAPE**. Ему передают символ, например, восклицательный знак. В регулярном выражении этот восклицательный знак сообщает: «Символ, который идёт после меня не относится к регулярному выражению. Это подстрока, которую нужно найти». Вот фрагмент кода, который найдёт все строки, заканчивающиеся на знак процента **%** в таблице:

```
название_столбца LIKE '%!%' ESCAPE '!'
--найдёт все строки заканчивающиеся на %
```

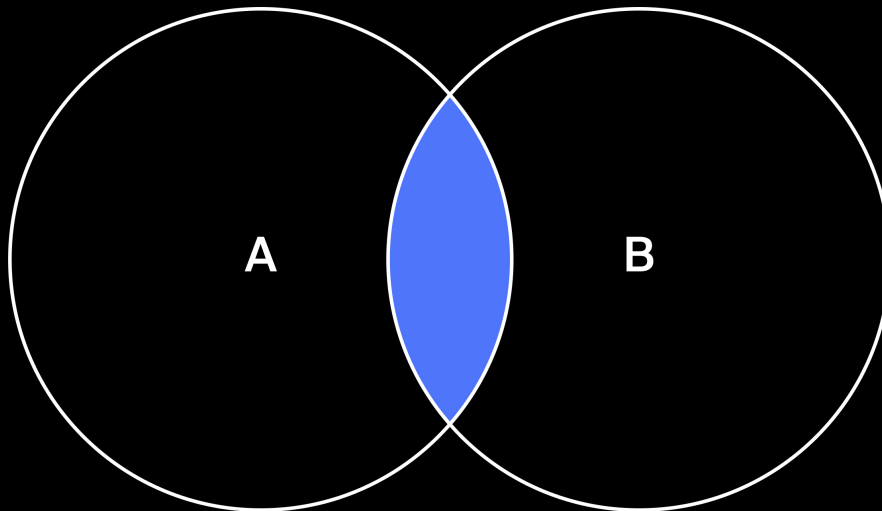
## JOIN. INNER JOIN.

Все данные редко хранят в одной таблице. Обычно таблицы приходится соединять.

Для этого есть оператор **JOIN**. «Джойнят» таблицы двумя способами: **INNER** и **OUTER**.

Соединение **INNER** выдаёт строки строго на пересечении двух таблиц:

## inner join



OUTER получает полные данные первой таблицы и к ним добавляет данные на пересечении со второй таблицей:

## outer join



## INNER JOIN

INNER JOIN формирует выборку только из тех данных, условие для присоединения которых выполняется. От порядка присоединения таблиц

результат не изменится.

Пример формата запроса с INNER JOIN:

```
SELECT --перечисляем 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
```

Разберём синтаксис подробнее:

- INNER JOIN — название способа соединения, после него указывают имя таблицы, с которой нужно соединить таблицу из блока FROM;
- ON открывает условие для присоединения: `TABLE_2.поле_1 = TABLE_1.поле_2`. То есть, соединяют только те строки таблиц, которые соответствуют условию. В нашем случае, равенству значений полей.

Так как поля в разных таблицах могут быть названы одинаково, к ним обращаются с указанием имени таблицы. Сперва идёт название таблицы, а потом имя её поля: `TABLE_1.поле_1`.

## Внешнее объединение таблиц. LEFT JOIN

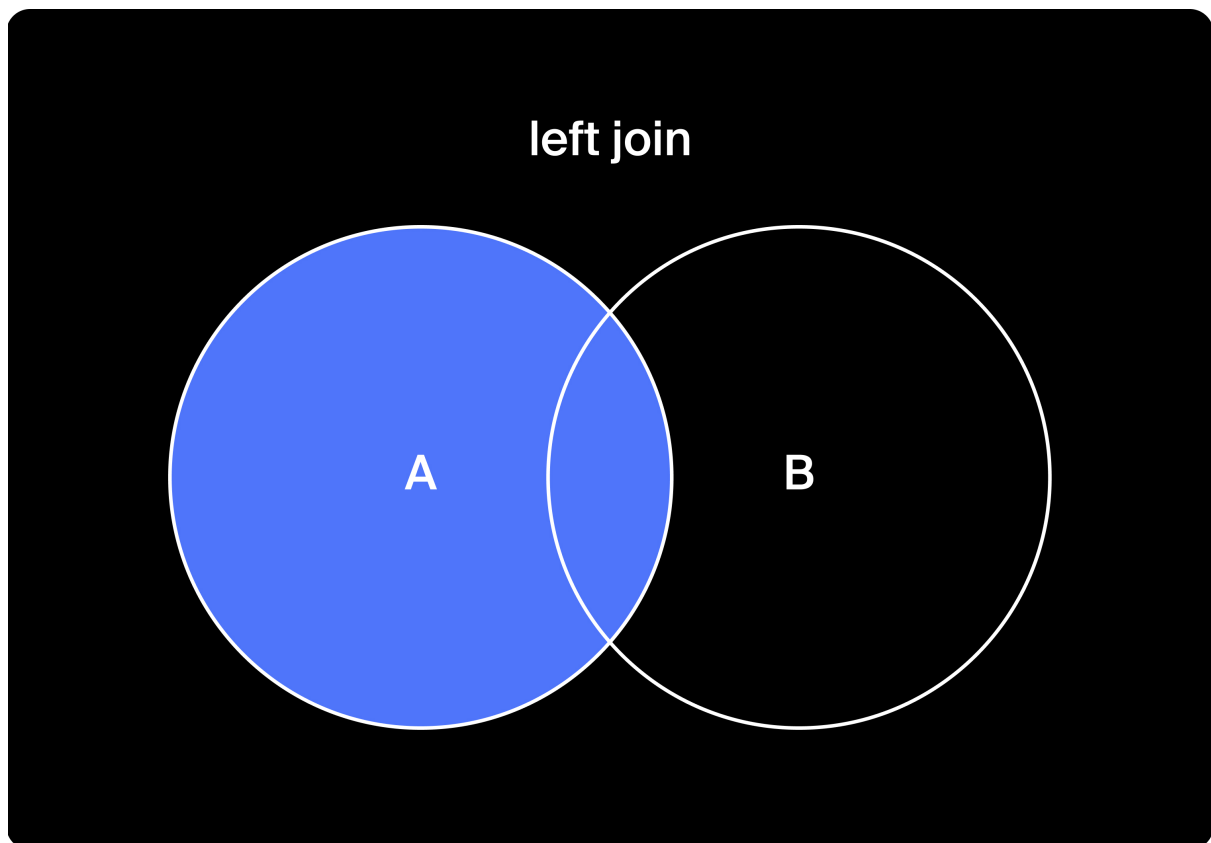
OUTER JOIN, или внешнее соединение таблиц выполняют двумя способами:

- **LEFT OUTER JOIN**
- **RIGHT OUTER JOIN**

Назовём способы объединения кратко: **LEFT JOIN** и **RIGHT JOIN**.

LEFT JOIN возьмёт все данные *левой* таблицы и строки на пересечении левой и правой, которые удовлетворяют условию присоединения. RIGHT JOIN, наоборот, возьмёт всю *правую* таблицу, и строки на пересечении с левой, соответствующие условию.





Пример формата запроса с LEFT JOIN:

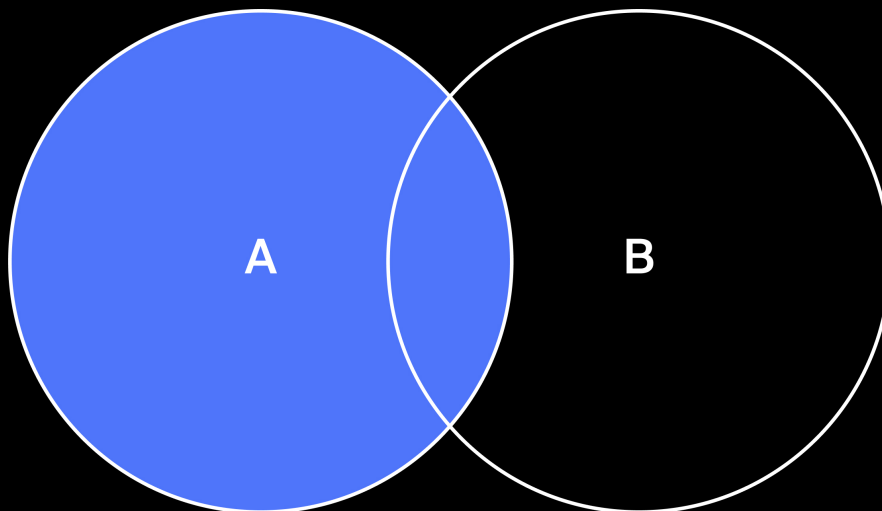
```
SELECT
  TABLE_1.поле_1 AS поле_1,
  TABLE_1.поле_2 AS поле_2,
  ...
  TABLE_2.поле_n AS поле_n
FROM
  TABLE_1
LEFT JOIN TABLE_2 ON TABLE_2.поле = TABLE_1.поле
```

Как и в запросах с INNER JOIN, название таблицы указывают для каждого поля. Обратите внимание, что в отличие от INNER JOIN перестановка таблиц меняет результат.

## Внешнее объединение таблиц. RIGHT JOIN

RIGHT JOIN — брат-близнец LEFT JOIN. Вот только в отличие от него, в результат своей работы берёт всю *правую* таблицу, и строки на пересечении с левой, если они соответствуют условию.

## right join



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 присоединяем вторую, а потом третью.

## Объединение запросов

Операторы **UNION** и **UNION ALL** «склеивают» данные из таблиц.

Синтаксис операторов выглядит так:

```
SELECT  
    название_столбца_1  
FROM  
    таблица_1  
UNION -- ( или UNION ALL )  
SELECT  
    название_столбца_1  
FROM  
    таблица_2
```

Два запроса SELECT - FROM разделяет команда UNION.

Условия, которые нужно соблюдать при «склеивании» данных:

- Данные из первой таблицы должны совпадать с данными из второй таблицы по количеству выбранных столбцов и типу данных;
- Поля первой таблицы нужно указывать в том же порядке, что и поля второй таблицы.

UNION «склеивает» данные так, что дублирующиеся строки не попадают в результат.

