

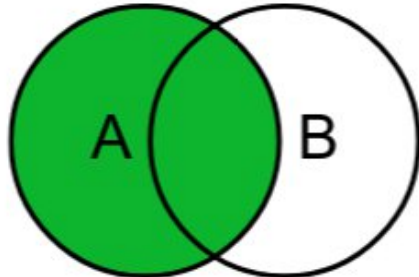
Лекция 6.

Язык SQL - продолжение

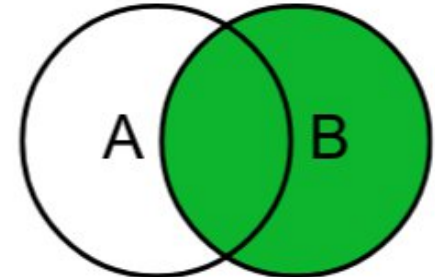
Соединение таблиц при выборке

SELECT: соединение таблиц

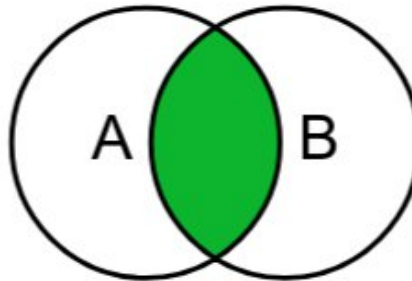
SQL JOINS



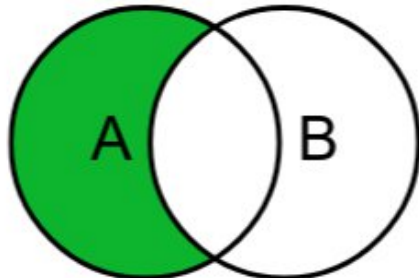
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



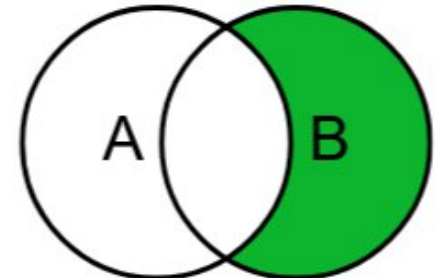
```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



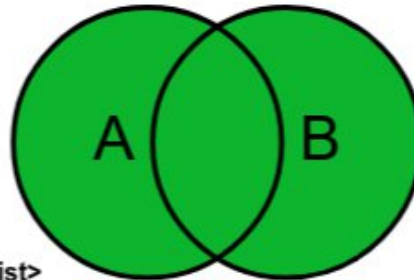
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



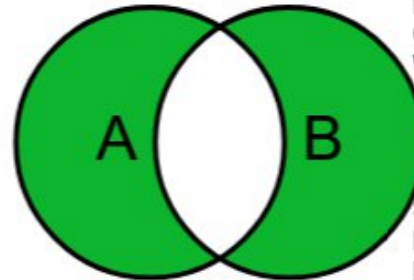
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```

Соединение таблиц

Соединение по равенству (внутреннее соединение)

Таблица E: студенты

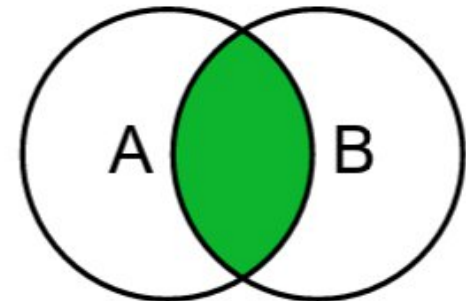
ID	Имя
1	Иван
2	Мария

Таблица F: оценки

ID	Предмет	Оценка
1	Математика	5
2	Физика	4

E ⋈ F:

ID	Имя	Предмет	Оценка
1	Иван	Математика	5
2	Мария	Физика	4



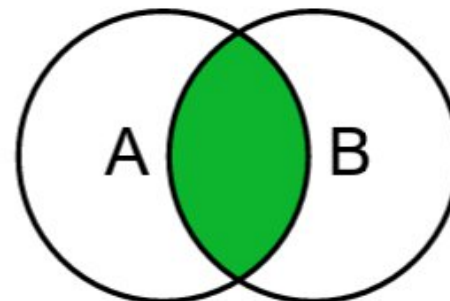
Внутреннее соединение

Условия соединения и фильтрация внутри WHERE

```
SELECT customer.name, vendor.name, vendor.city  
FROM vendor, customer  
WHERE vendor.city=customer.city AND vendor.percent > 11
```

Условие соединения внутри FROM, условие фильтрации внутри WHERE

```
SELECT customer.name, vendor.name, vendor.city  
FROM vendor  
    INNER JOIN customer ON vendor.city=customer.city  
WHERE vendor.percent > 11
```

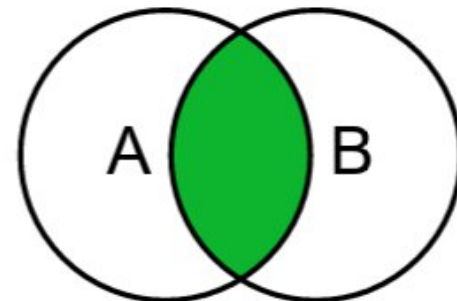


Внутреннее соединение таблиц

- В ранних версиях SQL (до стандарта SQL-92) не существовало синтаксиса `JOIN`.
- Соединения делались через декартово произведение + фильтрацию в `WHERE`:

```
sql
1 SELECT * FROM a, b WHERE a.id = b.id;
```

- Начиная с SQL-92, был введён явный синтаксис `JOIN`, чтобы отделить логику соединения от логики фильтрации.



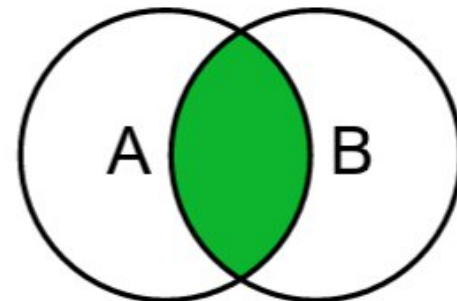
Внутреннее соединение таблиц

× Старый стиль (запутанно и опасно):

```
sql
```

```
1 SELECT *  
2 FROM users, orders, products  
3 WHERE users.id = orders.user_id  
4     AND orders.product_id = products.id  
5     AND products.name = 'Laptop';
```

→ Трудно понять, какие таблицы связаны, а что — фильтр.



Внутреннее соединение таблиц

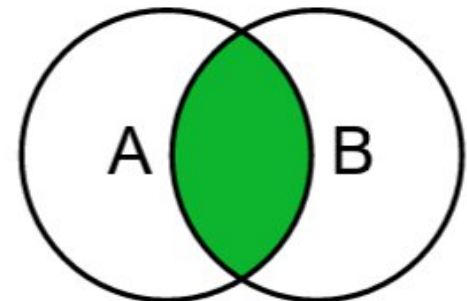
✓ Новый стиль (понятно и безопасно):

```
sql
```

```
1 SELECT *  
2 FROM users  
3 INNER JOIN orders ON users.id = orders.user_id  
4 INNER JOIN products ON orders.product_id = products.id  
5 WHERE products.name = 'Laptop';
```

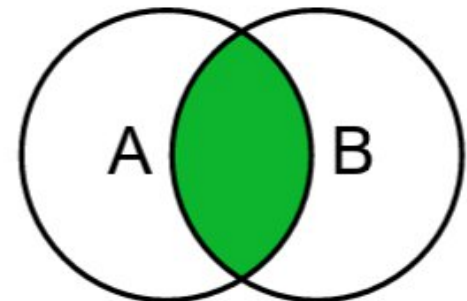
→ Чётко видно:

- `users` соединяется с `orders` по `user_id`,
- `orders` соединяется с `products` по `product_id`,
- А `products.name = 'Laptop'` — это **фильтр**, а не условие соединения.



Внутреннее соединение таблиц

- Во всех современных СУБД (PostgreSQL, MySQL, SQL Server, Oracle, SQLite) оптимизатор преобразует оба варианта в один и тот же план выполнения.
 - То есть разницы в скорости нет.
 - Но явный `JOIN` помогает оптимизатору лучше понимать намерения программиста.
- ✓ Всегда используйте явный синтаксис `JOIN` :
- Это стандарт де-факто в индустрии.
 - Код становится понятнее, особенно в сложных запросах.



Соединение трёх и более таблиц

```
SELECT customer.name 'Покупатель', vendor.name 'Продавец', vendor.city,  
`order`.order_date, `order`.summa  
FROM vendor  
  INNER JOIN customer  
    ON vendor.city=customer.city  
  INNER JOIN `order`  
    ON `order`.customer_id=customer.id;
```

Соединение трёх и более таблиц

`INNER JOIN` — это коммутативная и ассоциативная операция в реляционной алгебре. Это означает:

- `A INNER JOIN B` \equiv `B INNER JOIN A`
- `(A INNER JOIN B) INNER JOIN C` \equiv `A INNER JOIN (B INNER JOIN C)`

Поэтому **логически** (т.е. какие строки попадут в результат) — порядок не важен, **если условия соединения корректны и однозначны**.

Пример:

```
sql
1  -- Вариант 1
2  SELECT * FROM users
3  INNER JOIN orders ON users.id = orders.user_id
4  INNER JOIN products ON orders.product_id = products.id;
5
6  -- Вариант 2
7  SELECT * FROM products
8  INNER JOIN orders ON products.id = orders.product_id
9  INNER JOIN users ON orders.user_id = users.id;
```

Оба запроса вернут **один и тот же набор строк** (возможно, в разном порядке, но это не влияет на содержание).

Самообъединение

Найти все пары покупателей, имеющих одинаковый рейтинг

```
SELECT a.name AS Customer1, b.name AS Customer2, a.rating  
FROM customer a, customer b  
WHERE a.rating=b.rating
```

Соединение с результатом подзапроса

Вывести имена покупателей и суммы их заказов, но только для тех, у кого более двух заказов.

```
SELECT
    c.name AS customer_name,
    order_stats.total_sum
FROM customers c
INNER JOIN (
    SELECT
        customer_id,
        SUM(summa) AS total_sum,
        COUNT(*) AS order_count
    FROM orders
    GROUP BY customer_id
    HAVING COUNT(*) > 2
) AS order_stats ON c.id = order_stats.customer_id;
```

Соединение с результатом подзапроса

Вывести имена покупателей и суммы их заказов, но только для тех, у кого более двух заказов.

```
SELECT
    c.name AS customer_name,
    SUM(o.summa) AS total_sum
FROM customers c
INNER JOIN orders o ON c.id = o.customer_id
GROUP BY c.id, c.name
HAVING COUNT(o.id) > 2;
```

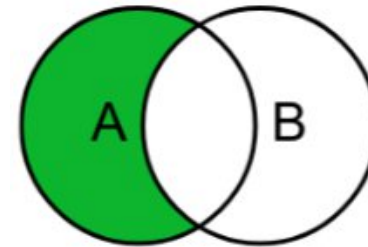
Да, тот же результат можно получить без подзапроса, и агрегацию можно (и нужно!) применять непосредственно к результату `INNER JOIN`. На самом деле, такой подход часто проще, короче и эффективнее.

Внешнее соединение таблиц

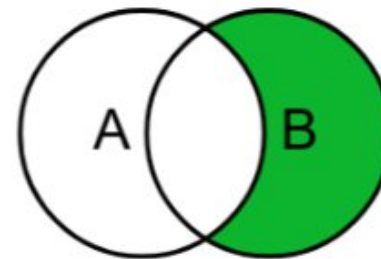
В отличие от внутренних объединений, которые связывают строки двух таблиц, внешние объединения включают в результат также строки, не имеющие пар.

Вывести все пары Продавец-Покупатель (в том числе продавцов, не имеющих покупателей)

```
SELECT vendor.name, customer.name  
FROM vendor LEFT OUTER JOIN customer  
ON vendor.id = customer.vendor_id
```



```
SELECT vendor.name, customer.name  
FROM customer RIGHT OUTER JOIN vendor  
ON vendor.id = customer.vendor_id
```



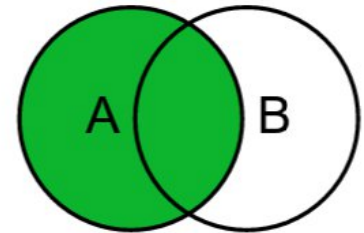
SELECT: соединение нескольких таблиц

Внешнее соединение

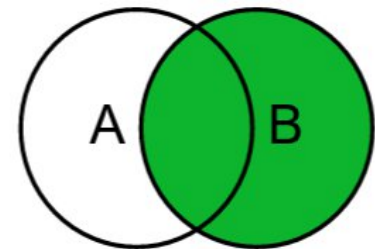
В отличие от внутренних объединений, которые связывают строки двух таблиц, внешние объединения включают в результат также строки, не имеющие пар.

Вывести все пары Продавец-Покупатель (в том числе продавцов, не имеющих покупателей)

```
SELECT vendor.name, customer.name  
FROM vendor LEFT OUTER JOIN customer  
ON vendor.id = customer.vendor_id
```



```
SELECT vendor.name, customer.name  
FROM customer RIGHT OUTER JOIN vendor  
ON vendor.id = customer.vendor_id
```



Реляционные связи между таблицами с помощью внешних ключей

Внешний ключ

Внешний ключ (**foreign key**) — это атрибут (или набор атрибутов) в одной таблице реляционной базы данных, который ссылается на **первичный ключ** (**primary key**) другой таблицы (или той же самой таблицы). Он обеспечивает целостность данных и связи между таблицами, реализуя концепцию реляционных связей.

Для чего нужен внешний ключ

1. Обеспечение ссылочной целостности (referential integrity)

Гарантирует, что значение внешнего ключа либо:

- соответствует существующему значению первичного ключа в связанной таблице,
- либо является `NULL` (если разрешено).

2. Создание связей между таблицами

Позволяет моделировать отношения: один-ко-многим, один-к-одному, многие-ко-многим (через связующую таблицу).

Для чего нужен внешний ключ

3. Поддержка каскадных операций

Например, при удалении родительской записи можно автоматически удалять или обновлять дочерние записи (`ON DELETE CASCADE` , `ON UPDATE CASCADE`).

4. Упрощение запросов с JOIN

Внешние ключи делают связи явными, что помогает СУБД оптимизировать запросы и строить планы выполнения.

Внешний ключ - примеры

Пример 1: Связь "один ко многим" — Клиенты и Заказы

```
sql
1  -- Таблица клиентов
2  CREATE TABLE Clients (
3      client_id INT PRIMARY KEY,
4      name VARCHAR(100)
5  );
6
7  -- Таблица заказов
8  CREATE TABLE Orders (
9      order_id INT PRIMARY KEY,
10     order_date DATE,
11     client_id INT, -- внешний ключ
12
13     FOREIGN KEY (client_id) REFERENCES Clients(client_id)
14         ON DELETE CASCADE -- при удалении клиента удаляются его заказы
15         ON UPDATE CASCADE -- при изменении client_id обновляются и в Orders
16 );
```

"Здесь `client_id` в таблице `Orders` — внешний ключ, ссылающийся на `client_id` в `Clients` .

Это означает: *каждый заказ принадлежит какому-то клиенту.*"

Внешний ключ - примеры

Пример 2: Самосвязь (рекурсивная связь) — Сотрудники и их руководители

sql

```
1 CREATE TABLE Employees (  
2     emp_id INT PRIMARY KEY,  
3     name VARCHAR(100),  
4     manager_id INT NULL, -- может быть NULL (например, у генерального директора)  
5  
6     FOREIGN KEY (manager_id) REFERENCES Employees(emp_id)  
7 );
```

“Здесь `manager_id` ссылается на `emp_id` в той же таблице.

Это позволяет моделировать иерархию: сотрудник → его руководитель → руководитель руководителя и т.д. ”

Пример 3: Связь "многие ко многим" — Студенты и Курсы через связующую таблицу

```
sql
1 v CREATE TABLE Students (
2     student_id INT PRIMARY KEY,
3     name VARCHAR(100)
4 );
5
6 v CREATE TABLE Courses (
7     course_id INT PRIMARY KEY,
8     title VARCHAR(200)
9 );
10
11 -- Связующая таблица
12 v CREATE TABLE Enrollments (
13     student_id INT,
14     course_id INT,
15     enrollment_date DATE,
16
17     PRIMARY KEY (student_id, course_id),
18
19     FOREIGN KEY (student_id) REFERENCES Students(student_id)
20         ON DELETE CASCADE,
21
22     FOREIGN KEY (course_id) REFERENCES Courses(course_id)
23         ON DELETE CASCADE
24 );
```

Внешний ключ

При создании таблицы с помощью оператора `CREATE TABLE` в SQL внешний ключ можно задать **двумя способами**:

1. **Внутри определения столбца (inline / column-level constraint)**
2. **Отдельно после определения всех столбцов (table-level constraint)**

Внешний ключ на уровне столбца

Задаётся сразу после объявления столбца, который является внешним ключом.

✅ Подходит, когда:

- Внешний ключ состоит из **одного столбца**.

📌 Синтаксис:

```
sql
1 v CREATE TABLE table_name (
2     column_name data_type REFERENCES referenced_table (referenced_column),
3     ...
4 );
```

📄 Пример:

```
sql
1 v CREATE TABLE Orders (
2     order_id INT PRIMARY KEY,
3     client_id INT REFERENCES Clients(client_id), -- inline FK
4     order_date DATE
5 );
```

Внешний ключ на уровне таблицы

Задаётся **отдельной строкой** в теле `CREATE TABLE`, после всех столбцов.

Подходит всегда, особенно когда:

- Внешний ключ состоит из **нескольких столбцов** (составной ключ),
- Вы хотите явно дать имя ограничению (constraint),
- Хотите использовать дополнительные опции (`ON DELETE`, `ON UPDATE` и т.д.) в более читаемом виде.

Внешний ключ на уровне таблицы

Пример 1 (простой):

```
sql
1 CREATE TABLE Orders (
2     order_id INT PRIMARY KEY,
3     client_id INT,
4     order_date DATE,
5     FOREIGN KEY (client_id) REFERENCES Clients(client_id)
6         ON DELETE CASCADE
7         ON UPDATE CASCADE
8 );
```

Пример 2 (составной внешний ключ):

```
sql
1 CREATE TABLE OrderDetails (
2     order_id INT,
3     product_id INT,
4     quantity INT,
5     PRIMARY KEY (order_id, product_id),
6     FOREIGN KEY (order_id, product_id)
7         REFERENCES ValidCombinations(order_id, product_id)
8 );
```

“Здесь внешний ключ состоит из двух столбцов — задать его на уровне столбца **НЕВОЗМОЖНО**, только на уровне таблицы.”

Каскадные операции

Зачем нужны каскадные операции

Каскадные операции (или **каскадное поведение**, *cascading actions*) в реляционных базах данных — это механизм автоматического выполнения действий в дочерних (зависимых) таблицах при изменении или удалении записей в родительской (основной) таблице, на которую ссылаются внешние ключи.

Они настраиваются при определении внешнего ключа с помощью опций `ON DELETE` и `ON UPDATE`.

Они обеспечивают:

- **Целостность данных** (никаких «висячих» ссылок),
- **Автоматизацию** сопровождения связанных записей,
- **Предсказуемое поведение** при изменениях в родительской таблице.

Каскадные операции при удалении

1. **ON DELETE** — что делать при удалении записи в родительской таблице

ВАРИАНТ	ПОВЕДЕНИЕ	↓
CASCADE	Удалить все связанные записи в дочерней таблице.	
SET NULL	Установить значение внешнего ключа в NULL (только если столбец допускает NULL).	
SET DEFAULT	Установить значение по умолчанию (если оно задано для столбца).	
RESTRICT / NO ACTION	Запретить удаление, если существуют связанные записи. (Это поведение по умолчанию во многих СУБД.)	

Каскадные операции при удалении

2. **ON UPDATE** — что делать при изменении значения первичного ключа в родительской таблице

ВАРИАНТ	ПОВЕДЕНИЕ
CASCADE	Автоматически обновить значение внешнего ключа в дочерней таблице.
SET NULL	Установить внешний ключ в NULL .
SET DEFAULT	Установить значение по умолчанию.
RESTRICT / NO ACTION	Запретить изменение первичного ключа, если есть связанные записи.

Каскадные операции - примеры

Пример 1: Удаление клиента → удалить все его заказы

```
sql
1 CREATE TABLE Clients (
2     id INT PRIMARY KEY,
3     name VARCHAR(100)
4 );
5
6 CREATE TABLE Orders (
7     id INT PRIMARY KEY,
8     client_id INT,
9     amount DECIMAL(10,2),
10
11     FOREIGN KEY (client_id) REFERENCES Clients(id)
12         ON DELETE CASCADE -- при удалении клиента – удаляются его заказы
13 );
```

Каскадные операции - примеры

Пример 2: Удаление категории → обнулить ссылку в товарах

```
sql
1 v CREATE TABLE Categories (
2     id INT PRIMARY KEY,
3     name VARCHAR(50)
4 );
5
6 v CREATE TABLE Products (
7     id INT PRIMARY KEY,
8     name VARCHAR(100),
9     category_id INT NULL, -- допускает NULL
10
11     FOREIGN KEY (category_id) REFERENCES Categories(id)
12         ON DELETE SET NULL -- если категория удалена – товар остаётся без категории
13 );
```

Каскадные операции - примеры

Пример 3: Запрет удаления, если есть зависимые записи

```
sql
1 v CREATE TABLE Authors (
2     id INT PRIMARY KEY,
3     name VARCHAR(100)
4 );
5
6 v CREATE TABLE Books (
7     id INT PRIMARY KEY,
8     title VARCHAR(200),
9     author_id INT,
10
11     FOREIGN KEY (author_id) REFERENCES Authors(id)
12         ON DELETE RESTRICT -- или просто не указывать ON DELETE (поведение по умолчанию)
13 );
```

Подзапросы

Подзапросы

Подзапрос - это запрос, помещённый (в круглых скобках) в другую инструкцию SQL.

- **Некоррелированные (вложенные)** подзапросы. Выполняются первыми, независимо от содержащего запроса.
- **Коррелированные (связанные)** подзапросы. Выполняются в составе содержащего запроса, ссылаются на один или несколько столбцов из содержащей инструкции.

Вложенные (некоррелированные) запросы

Внутри HAVING можно применять подзапросы, которые не дают множества значений.

Найти количество покупателей с рейтингом, превышающим среднее значение для покупателей из Саранска

```
SELECT rating, COUNT(DISTINCT id) FROM customer  
GROUP BY rating  
HAVING rating > (SELECT AVG(rating)  
FROM customer WHERE city="Саранск")
```

Связанные (коррелированные) подзапросы

Найти всех покупателей, сделавших заказы 10.12.2016

```
SELECT * FROM customer a
WHERE "2016-12-10" IN
  (SELECT order_date FROM order b WHERE a.id = b.customer_id)
```

Внутренний запрос должен выполняться отдельно для каждой строки внешнего запроса.

1. Выбирается текущая строка-кандидат из таблицы, указанной во внешнем запросе.
2. Значение этой строки сохраняется в псевдониме из FROM внешнего запроса.
3. Выполняется подзапрос. Каждый раз, когда встречается псевдоним для внешнего запроса, его значение применяется к текущей строке-кандидату (внешней ссылке).
4. Оценивается предикат внешнего запроса на основе подзапроса, выполненного на шаге 3 (будет ли строка-кандидат включена в выходные данные).
5. Процедура повторяется для следующей строки-кандидата.

Связанные (коррелированные) подзапросы

1. Найдем имена и номера всех продавцов, имеющих более одного покупателя

```
SELECT id, name  
FROM vendor main  
WHERE 1 < (SELECT COUNT(*) FROM customer WHERE vendor_id=main.id)
```

2. Найдем все заказы, сумма в которых превышает среднюю сумму заказа для данного покупателя

```
SELECT *  
FROM order a  
WHERE сумма > (SELECT AVG(сумма) FROM `order` b  
WHERE a.customer_id = b.customer_id)
```

Подзапросы

Могут использоваться:

- В разделе WHERE операторов SELECT, UPDATE, DELETE.
- В разделе SET оператора UPDATE.
- В разделе VALUES оператора INSERT. Должны быть некоррелированными скалярными.
- В разделе FROM оператора SELECT. Должны быть некоррелированными.
- В разделе HAVING оператора SELECT ... FROM ... GROUP BY ...
- В разделе ORDER BY оператора SELECT. Подзапрос должен быть скалярным.

Количество возвращаемых подзапросом строк и столбцов

Подзапрос возвращает:

- **Одна строка, один столбец.** Скалярный подзапрос, можно использовать в условиях равенства.
- **Несколько строк, один столбец.** В проверках на включение в множество.
- **Несколько столбцов.** В проверках на включение в множество.

Количество возвращаемых подзапросом строк и столбцов

ID	NAME	DEPARTMENT	SALARY	⬇
1	Иван Петров	IT	90000	
2	Мария Сидорова	IT	95000	
3	Алексей Иванов	IT	85000	
4	Ольга Кузнецова	HR	60000	
5	Дмитрий Смирнов	HR	62000	
6	Елена Попова	Finance	75000	
7	Сергей Васильев	Finance	78000	
8	Анна Морозова	Marketing	68000	
9	Павел Новиков	Marketing	70000	
10	Татьяна Лебедева	Sales	65000	

Количество возвращаемых подзапросом строк и столбцов

1 Скалярный подзапрос

Одна строка, один столбец

Можно использовать вместо значения — например, в `WHERE`, `SELECT`, `HAVING`.

✓ Пример: Найти сотрудников с зарплатой выше средней по компании

```
sql
1 SELECT name, salary
2 FROM employees
3 WHERE salary > (SELECT AVG(salary) FROM employees);
```

- Подзапрос `(SELECT AVG(salary) FROM employees)` возвращает **одно значение** (например, `74800`).
- Это **скалярный подзапрос** — его результат можно сравнивать через `>`, `=`, `<` и т.д.

⚠ Если подзапрос вернёт **0 строк**, результат будет `NULL`, и условие не выполнится.

⚠ Если вернёт **более одной строки** — ошибка!

Количество возвращаемых подзапросом строк и столбцов

2 Подзапрос с несколькими строками, один столбец

Несколько строк, один столбец

Используется с операторами: `IN`, `NOT IN`, `ANY`, `ALL`.

✓ Пример: Найти сотрудников из отделов, где максимальная зарплата > 90000

```
sql
1 SELECT name, department
2 FROM employees
3 WHERE department IN (
4     SELECT department
5     FROM employees
6     GROUP BY department
7     HAVING MAX(salary) > 90000
8 );
```

- Внутренний подзапрос возвращает список отделов (один столбец, несколько строк):
→ `[IT]` (т.к. в IT макс. зарплата = 95000)
- Внешний запрос выбирает всех из этих отделов.

“ ♦ Используем `IN`, потому что результат — множество значений одного типа. ”

Несколько строк, несколько столбцов

Используется с `IN` / `NOT IN`, но **в виде кортежей** (пар, троек и т.д.).

✅ **Пример:** Найти сотрудников, чья комбинация (department, salary) совпадает с топ-1 по зарплате в каждом отделе

Сначала найдём топ-зарплаты по отделам:

```
sql
1  -- Подзапрос: (department, max_salary)
2  v SELECT department, MAX(salary) AS max_sal
3     FROM employees
4     GROUP BY department
```

Теперь используем это в основном запросе:

```
sql
1  v SELECT name, department, salary
2     FROM employees
3     WHERE (department, salary) IN (
4         SELECT department, MAX(salary)
5         FROM employees
6         GROUP BY department
7     );
```

- Подзапрос возвращает **пары:**

`(IT, 95000)` , `(HR, 62000)` , `(Finance, 78000)` , `(Marketing, 70000)` , `(Sales, 65000)`

- Основной запрос ищет строки, где **оба значения совпадают одновременно.**

Исходный запрос с подзапросом и кортежами:

```
sql
1 SELECT name, department, salary
2 FROM employees
3 WHERE (department, salary) IN (
4     SELECT department, MAX(salary)
5     FROM employees
6     GROUP BY department
7 );
```

Некоторые СУБД (например, **Microsoft SQL Server**) не поддерживают синтаксис `(col1, col2) IN (...)`. В таких случаях удобно использовать **JOIN**.

✓ Переписываем через **JOIN**

```
sql
1 SELECT e.name, e.department, e.salary
2 FROM employees e
3 JOIN (
4     SELECT department, MAX(salary) AS max_salary
5     FROM employees
6     GROUP BY department
7 ) dept_max
8 ON e.department = dept_max.department
9 AND e.salary = dept_max.max_salary;
```

Подзапросы в разделе FROM

Подзапрос выступает в качестве источника данных (формирует временную таблицу)

Пример. Для всех покупателей определить общее количество и сумму купленного товара, в зависимости от суммы отнести покупателя к одной из трех групп: «Мало» (до 100 рублей), «Средне» (от 100 до 500 рублей), «Много» (более 500 рублей).

Подзапросы в разделе ORDER BY

Вывести фамилии покупателей, отсортированные по количеству их покупок.

```
SELECT c.name  
FROM customer c  
ORDER BY  
(SELECT count(*) FROM `order` ord  
WHERE c.id=ord.customer_id) DESC;
```

Подзапросы в инструкции INSERT

Внести в таблицу order запись о покупке, если известны фамилии продавца и покупателя.

```
INSERT INTO `order` (summa, order_date, customer_id, vendor_id)
VALUES (200.25, '2021-09-21', (SELECT id FROM customer WHERE
name='Ли'), (SELECT id FROM vendor WHERE name='Петров'));
```

Подзапросы - итоги

- Либо не зависят от содержащего запроса, либо ссылаются на столбцы из содержащей инструкции.
- Используются в условиях WHERE в операторах сравнения и операторах IN, NOT IN, EXISTS, NOT EXISTS.
- Могут применяться в инструкциях SELECT, INSERT, UPDATE, DELETE.
- Могут создавать результирующие наборы, которые можно соединять в запросе с другими таблицами и подзапросами.
- Могут использоваться для генерации значений для заполнения таблицы или столбцов в результирующем наборе запроса.
- Используются в предложениях SELECT, FROM, WHERE, HAVING и ORDER BY.

Подзапросы - итоги

- **Некоррелированные подзапросы** — безопасны и эффективны.
- **Коррелированные подзапросы** — мощный инструмент, но их стоит использовать **только при необходимости**, когда альтернативы не подходят.
- В большинстве случаев их можно заменить на **JOIN'ы** или **оконные функции**, что улучшит производительность и читаемость кода.

📄 Структура таблицы `employees`

ID	NAME	DEPARTMENT	SALARY
1	Иван Петров	IT	90000
2	Мария Сидорова	IT	95000
3	Алексей Иванов	IT	85000
4	Ольга Кузнецова	HR	60000
5	Дмитрий Смирнов	HR	62000
6	Елена Попова	Finance	75000
7	Сергей Васильев	Finance	78000
8	Анна Морозова	Marketing	68000
9	Павел Новиков	Marketing	70000
10	Татьяна Лебедева	Sales	65000

1. Некоррелированный подзапрос

sql



```
1 v SELECT name
2   FROM employees
3  WHERE salary > (SELECT AVG(salary) FROM employees);
```

- Средняя зарплата по всей компании:

$$(90000 + 95000 + 85000 + 60000 + 62000 + 75000 + 78000 + 68000 + 70000 + 65000) / 10 = 74800$$

- Результат: сотрудники с зарплатой > 74800 →

**Иван Петров, Мария Сидорова, Алексей Иванов, Елена Попова, Сергей Васильев,
Анна Морозова, Павел Новиков**

2. Коррелированный подзапрос

```
sql
1 v SELECT e1.name
2 FROM employees e1
3 WHERE e1.salary > (
4     SELECT AVG(e2.salary)
5     FROM employees e2
6     WHERE e2.department = e1.department
7 );
```

- Средние по отделам:
 - **IT:** $(90000 + 95000 + 85000) / 3 = 90000$ → выше среднего: **Мария Сидорова** (95000)
 - **HR:** $(60000 + 62000) / 2 = 61000$ → выше среднего: **Дмитрий Смирнов** (62000)
 - **Finance:** $(75000 + 78000) / 2 = 76500$ → выше среднего: **Сергей Васильев** (78000)
 - **Marketing:** $(68000 + 70000) / 2 = 69000$ → выше среднего: **Павел Новиков** (70000)
 - **Sales:** только один сотрудник → среднее = 65000 → **никто не выше среднего**
- Итоговый результат:
Мария Сидорова, Дмитрий Смирнов, Сергей Васильев, Павел Новиков

✓ Альтернативы коррелированным подзапросам

1. Использование JOIN'ов:

```
sql
1 v SELECT e1.name
2   FROM employees e1
3   JOIN (
4     SELECT department, AVG(salary) AS avg_sal
5     FROM employees
6     GROUP BY department
7   ) dept_avg ON e1.department = dept_avg.department
8  WHERE e1.salary > dept_avg.avg_sal;
```

2. Оконные функции (если поддерживается СУБД):

```
sql
1 v SELECT name
2   FROM (
3     SELECT name, salary, department,
4            AVG(salary) OVER (PARTITION BY department) AS dept_avg
5     FROM employees
6   ) e
7  WHERE salary > dept_avg;
```

Эти подходы обычно **быстрее, масштабируемее** и **лучше оптимизируются**.

SELECT: ключевые слова ANY, ALL

Используются только совместно с подзапросами. В SQLite нет.

Если подзапросу предшествует ключевое слово **ALL**, условие сравнения считается выполненным, только когда оно выполняется для всех значений в результирующей столбце подзапроса.

Выбрать все заказы, сумма которых превосходит суммы всех заказов, сделанных 10.12.2016

```
SELECT * FROM `order`  
WHERE сумма > ALL (  
  SELECT сумма FROM order WHERE order_date="2016-12-10")
```