Сложные запросы и операции

1. Виды объединений таблиц  
**Суть:** Механизмы соединения таблиц по условию.

Основные типы JOIN:

1. **INNER JOIN** - возвращает только совпадающие строки
2. **LEFT JOIN** - все строки левой таблицы + совпадения справа
3. **RIGHT JOIN** - все строки правой таблицы + совпадения слева
4. **FULL JOIN** - все строки обеих таблиц
5. **CROSS JOIN** - декартово произведение (все комбинации)
6. **SELF JOIN** - соединение таблицы с самой собой

Специальные виды:

* **NATURAL JOIN** - автоматическое соединение по одинаковым именам столбцов
* **LATERAL JOIN** - подзапрос, который может ссылаться на предыдущие таблицы

2. Примеры объединений

INNER JOIN (пример):

sql

Copy

Download

SELECT o.order\_id, c.customer\_name

FROM orders o

INNER JOIN customers c ON o.customer\_id = c.customer\_id;

LEFT JOIN (пример):

sql

Copy

Download

SELECT d.department\_name, e.employee\_name

FROM departments d

LEFT JOIN employees e ON d.department\_id = e.department\_id;

CROSS JOIN (пример):

sql

Copy

Download

SELECT p.product\_name, s.size\_name

FROM products p

CROSS JOIN sizes s;

SELF JOIN (пример иерархии):

sql

Copy

Download

SELECT e1.name AS employee, e2.name AS manager

FROM employees e1

LEFT JOIN employees e2 ON e1.manager\_id = e2.employee\_id;

3. Круги Эйлера для визуализации JOIN  
**Суть:** Визуализация логики соединений через диаграммы пересечений.  
**Зачем:** Помогает понять, какие данные останутся после JOIN.

Визуальное представление типов соединений:

Copy

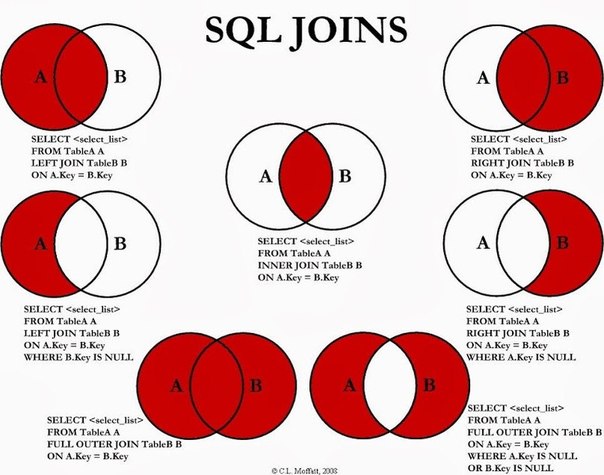
Download

INNER JOIN: [A]∩[B]

LEFT JOIN: [A]∪(A∩B)

RIGHT JOIN: [B]∪(A∩B)

FULL JOIN: [A]∪[B]

CROSS JOIN: [A]×[B]  


4. Вложенные запросы

Виды вложенных запросов:

1. В WHERE:

sql

Copy

Download

SELECT product\_name

FROM products

WHERE category\_id IN (SELECT category\_id FROM categories WHERE active = true);

1. В FROM:

sql

Copy

Download

SELECT avg\_order.avg\_price

FROM (SELECT AVG(price) as avg\_price FROM orders) as avg\_order;

1. В SELECT:

sql

Copy

Download

SELECT product\_id,

(SELECT COUNT(\*) FROM orders WHERE product\_id = p.product\_id) as order\_count

FROM products p;

1. С EXISTS:

sql

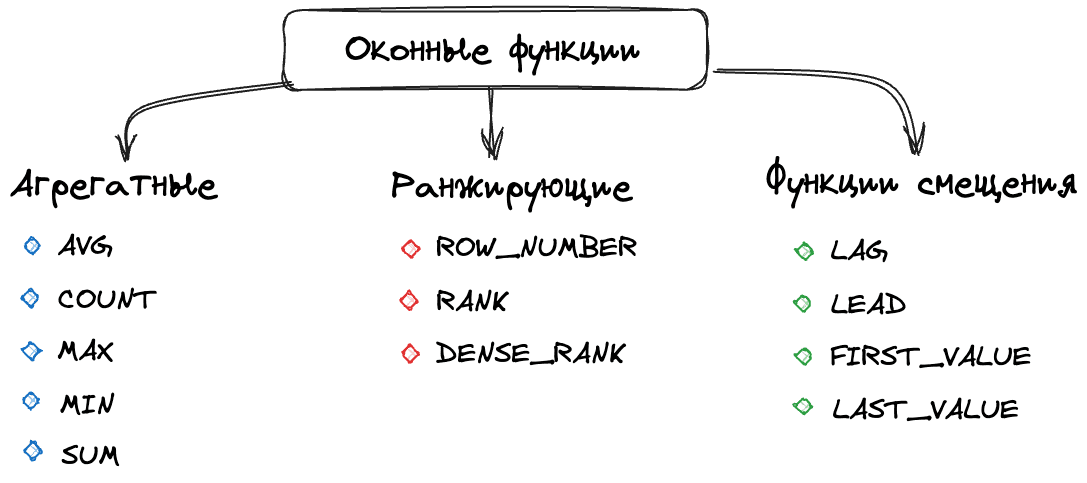
Copy

Download

SELECT customer\_id, name

FROM customers c

WHERE EXISTS (SELECT 1 FROM orders WHERE customer\_id = c.customer\_id);

5. Оконные функции  


Основные функции:

sql

Copy

Download

SELECT

product\_id,

product\_name,

price,

RANK() OVER (ORDER BY price DESC) as price\_rank,

AVG(price) OVER (PARTITION BY category\_id) as avg\_category\_price,

FIRST\_VALUE(product\_name) OVER (PARTITION BY category\_id ORDER BY price DESC) as top\_product

FROM products;

Ключевые элементы:

* PARTITION BY - аналог GROUP BY для оконных функций
* ORDER BY - сортировка внутри окна
* ROWS/RANGE - определение границ окна

6. Представления (Views)

Создание представления:

sql

Copy

Download

CREATE VIEW active\_customers AS

SELECT customer\_id, name, email

FROM customers

WHERE active = true AND last\_purchase\_date > CURRENT\_DATE - INTERVAL '1 year';

Материализованные представления:

sql

Copy

Download

CREATE MATERIALIZED VIEW monthly\_sales AS

SELECT

DATE\_TRUNC('month', order\_date) as month,

SUM(amount) as total\_sales

FROM orders

GROUP BY 1

REFRESH MATERIALIZED VIEW monthly\_sales;

**7. Обычные CTE (Common Table Expressions)**

**Суть:** Временные именованные результаты запроса, которые существуют только во время выполнения основного запроса.

**Особенности:**

* Определяются через WITH имя\_CTE AS (запрос)
* Улучшают читаемость, разбивая сложные запросы на логические блоки
* Могут использоваться несколько раз в одном запросе
* Не хранятся в базе (в отличие от представлений)

**Примерная структура:**

sql

Copy

Download

WITH временная\_таблица AS (

SELECT ... FROM ... WHERE ...

)

SELECT \* FROM временная\_таблица JOIN ...;

**Когда использовать:**

* Для упрощения сложных запросов
* Когда нужно многократно использовать один подзапрос
* Как альтернатива вложенным подзапросам

**2. Рекурсивные CTE**

**Суть:** Специальный тип CTE, который может ссылаться на самого себя, позволяя обрабатывать иерархические или рекурсивные структуры данных.

**Ключевые особенности:**

* Всегда содержит две части, объединенные UNION ALL:
  1. **Якорная часть (базовый случай)** - начальные данные
  2. **Рекурсивная часть** - обращение к самому CTE
* Требует ключевого слова RECURSIVE (в большинстве СУБД)
* Имеет ограничение глубины рекурсии (защита от бесконечных циклов)

**Примерная структура:**

sql

Copy

Download

WITH RECURSIVE иерархия AS (

-- Якорная часть (начальные строки)

SELECT id, parent\_id, name FROM table WHERE parent\_id IS NULL

UNION ALL

-- Рекурсивная часть

SELECT t.id, t.parent\_id, t.name

FROM table t

JOIN иерархия h ON t.parent\_id = h.id

)

SELECT \* FROM иерархия;

**Когда использовать:**

1. **Обработка иерархий:**
   * Организационные структуры
   * Деревья комментариев
   * Категории товаров с вложенностью
2. **Обход графов:**
   * Поиск путей между точками
   * Социальные связи (друзья друзей)
3. **Генерация последовательностей:**
   * Создание рядов дат
   * Генерация тестовых данных

**Сравнительная таблица**

| **Характеристика** | **Обычные CTE** | **Рекурсивные CTE** |
| --- | --- | --- |
| **Самоссылка** | Нет | Да (через UNION ALL) |
| **Сложность** | Линейные запросы | Рекурсивные алгоритмы |
| **Типичное применение** | Упрощение запросов | Иерархии, графы |
| **Производительность** | Быстрые | Могут быть ресурсоёмкими |
| **Синтаксис** | WITH имя AS (...) | WITH RECURSIVE имя AS (...) |

**Важные нюансы рекурсивных CTE**

1. **Условие остановки** - рекурсивная часть должна в какой-то момент перестать возвращать строки
2. **Ограничение глубины** - например, в PostgreSQL по умолчанию 100 итераций
3. **Производительность** - для больших иерархий могут потребоваться индексы

**Пример ограничения глубины (PostgreSQL):**

sql

Copy

Download

WITH RECURSIVE cte AS (...)

SELECT \* FROM cte

OPTION (MAXRECURSION 200); -- Явное указание глубины

**Вывод**

* **Обычные CTE** - инструмент для структурирования запросов
* **Рекурсивные CTE** - мощный механизм для работы с иерархиями, недоступный обычным SQL-запросам
* Оба типа помогают сделать код чище и логичнее, но требуют разного подхода к проектированию