SQL: Работа с JOIN

Основные виды соединений и примеры использования

Лазар В. И., Козлова Е. Р.

16 января 2025 г.

План занятия

Общее представление о JOIN

- **JOIN** — это операция в SQL, позволяющая объединять строки из двух (или более) таблиц на основе связанного столбца между ними.
- Используется для того, чтобы в одном запросе получать данные сразу из нескольких таблиц.
- Наиболее распространены следующие виды **JOIN**:
  - **INNER JOIN**
  - **LEFT (OUTER) JOIN**
  - **RIGHT (OUTER) JOIN**
  - **FULL (OUTER) JOIN**
  - **CROSS JOIN**
- Другие специфические варианты (например, `NATURAL JOIN`) реже применяются и не всегда поддерживаются.

INNER JOIN — соединение по совпадающим значениям

- **INNER JOIN** возвращает только те строки, у которых совпадают значения в связанных столбцах обеих таблиц.
- Если хотя бы в одной таблице нет соответствующей записи, то такая строка не попадёт в результирующий набор.

Синтаксис

```
SELECT t1.col, t2.col
FROM table1 AS t1
INNER JOIN table2 AS t2
    ON t1.key = t2.key;
```

Пример использования INNER JOIN

Предположим, у нас есть две таблицы:

- `orders (order_id, customer_id, amount)`
- `customers (customer_id, customer_name)`

Запрос:

```
SELECT c.customer_name, o.amount
FROM customers AS c
INNER JOIN orders AS o
    ON c.customer_id = o.customer_id
WHERE o.amount > 1000;
```

Описание:

- Выбирает имя покупателя и сумму заказа.
- Возвращает только те строки, где `customer_id` есть и в `customers`, и в `orders`, и сумма заказа > 1000.

LEFT (OUTER) JOIN — все строки из левой таблицы

- **LEFT JOIN** возвращает все строки из левой таблицы (указанной после `FROM`), даже если в правой таблице нет соответствующих значений.
- Для строк, у которых нет совпадающей записи в правой таблице, значения в столбцах правой таблицы будут **NULL**.

Синтаксис

```
SELECT t1.col, t2.col
FROM table1 AS t1
LEFT JOIN table2 AS t2
    ON t1.key = t2.key;
```

Пример использования LEFT JOIN

Те же таблицы:

- `orders (order_id, customer_id, amount)`
- `customers (customer_id, customer_name)`

Запрос:

```
SELECT c.customer_name, o.amount
FROM customers AS c
LEFT JOIN orders AS o
    ON c.customer_id = o.customer_id;
```

Описание:

- Возвращает все строки из `customers`, даже если в `orders` для данного `customer_id` нет записей.
- Если покупатель не сделал заказ, `amount` будет **NULL**.

RIGHT (OUTER) JOIN — все строки из правой таблицы

- **RIGHT JOIN** возвращает все строки из правой таблицы, даже если в левой таблице нет соответствующих значений.
- Аналогично **LEFT JOIN**, но фокус — на правой таблице.
- Не все СУБД (например, SQLite) поддерживают `RIGHT JOIN` напрямую.

Синтаксис

```
SELECT t1.col, t2.col
FROM table1 AS t1
RIGHT JOIN table2 AS t2
    ON t1.key = t2.key;
```

Пример использования RIGHT JOIN

Если `RIGHT JOIN` поддерживается:

```
SELECT o.order_id, c.customer_name
FROM customers AS c
RIGHT JOIN orders AS o
    ON c.customer_id = o.customer_id;
```

Описание:

- Возвращает все строки из `orders`.
- Если в `customers` нет совпадающей записи по `customer_id`, то `customer_name` будет **NULL**.

FULL (OUTER) JOIN — все строки из обеих таблиц

- **FULL JOIN** возвращает все строки из обеих таблиц, заполняя **NULL** там, где нет совпадающих значений.
- Объединяет в себе идею **LEFT JOIN** и **RIGHT JOIN**.
- Тоже не поддерживается во всех СУБД (в частности, в SQLite нет **FULL JOIN**).

Синтаксис

```
SELECT t1.col, t2.col
FROM table1 AS t1
FULL JOIN table2 AS t2
    ON t1.key = t2.key;
```

CROSS JOIN — декартово произведение

- **CROSS JOIN** возвращает декартово произведение строк: каждая строка левой таблицы «скрещивается» с каждой строкой правой.
- При **CROSS JOIN** обычно не указывается условие `ON`, либо оно игнорируется.

Синтаксис

```
SELECT t1.col, t2.col
FROM table1 AS t1
CROSS JOIN table2 AS t2;
```

Когда лучше использовать JOIN, а когда — вложенные запросы?

- **JOIN** часто предпочтительнее по производительности при больших объёмах данных.
- **Вложенные запросы** удобны, когда нужно получить промежуточные вычисления (например, выборку уникальных значений или агрегацию), и логичнее оформить это «внутри» основного запроса.
- При **JOIN** можно в одном запросе отобразить данные из нескольких таблиц в виде одной «плоской» структуры.
- СУБД обычно эффективно оптимизируют **JOIN**, особенно если правильно настроены индексы.

Пример: JOIN vs Subquery

Пусть нужно вывести имена клиентов, сделавших заказы дороже 1000.

Через вложенный запрос:

```
SELECT customer_name
FROM customers
WHERE customer_id IN (
    SELECT customer_id
    FROM orders
    WHERE amount > 1000
);
```

Через JOIN:

```
SELECT DISTINCT c.customer_name
FROM customers AS c
JOIN orders AS o
    ON c.customer_id = o.customer_id
WHERE o.amount > 1000;
```

Задания для factbook.db и chinook.db

factbook.db

- Напишите запрос, возвращающий для каждой страны следующие значения: название страны, население страны, городское население страны и процентное соотношение городского населения к общему
- Напишите запрос, возвращающий каждой страны название, количество городов в этой стране, столицу этой страны, население столицы и процент населения столицы от общего населения

chinook.db

- Напишите запрос, получающие следующие колонки из объединения таблиц: invoice_id=1, track_id, track_name, media_type_name, quantity и unit_price
- Напишите запрос, получающий информацию о названии альбома, имени испонителя и общем количестве проданных копий альбома
- Для каждого клиента выведите суммарное количество средств на его счетах. Полученные данные отсортируйте в порядке убывания средств на счетах.

Задания с Leetcode

Для выполнения этих заданий необходимо зарегистрироваться на сайте **leetcode.com**, далее перейти в раздел **SQL 50**.

- Обе задачи сложности Medium из раздела **Basic Joins**
- Первые 3 задачи (Easy level) из раздела **Advanced Select and Joins**, а также задача **Count Salary Categories**