

# Тема 1. Технология ORM. Миграции

## Основные термины и понятия

**ORM** (англ. Object-Relational Mapping, объектно-реляционное отображение, или преобразование) — технология, которая позволяет программисту взаимодействовать с базами данных с помощью синтаксических конструкций других языков программирования.

**Объектно-ориентированное программирование** (ООП) — это методология программирования, которая представляет программу в виде совокупности взаимодействующих объектов.

**Объект** — это набор данных (атрибутов или свойств) и функций (методов). Свойства — это характеристики объекта, а методы — действия, которые умеет выполнять объект.

**ORM-библиотека** — конкретная реализация технологии ORM.

## Преимущества и недостатки использования ORM

Преимущества:

- простота разработки;
- унификация доступа к разным СУБД;
- обеспечение безопасности.

Недостатки: сложность изучения; снижение скорости работы; ограниченные возможности и сложность отладки.

**Инструмент миграций** — механизм, который позволяет выполнять DDL-запросы без SQL, то есть даёт возможность разработчику менять структуру БД. Конкретные реализации представлены различными библиотеками миграций.

## Как разработчик применяет миграцию:

1. Вносит изменения в код.
2. Запускает процесс автогенерации миграций.
3. Проверяет созданную миграцию визуально.
4. Применяет миграцию.
5. Проверяет появившиеся изменения в базе.
6. Тестирует откат миграции.

## Тема 2. Общие подходы к оптимизации

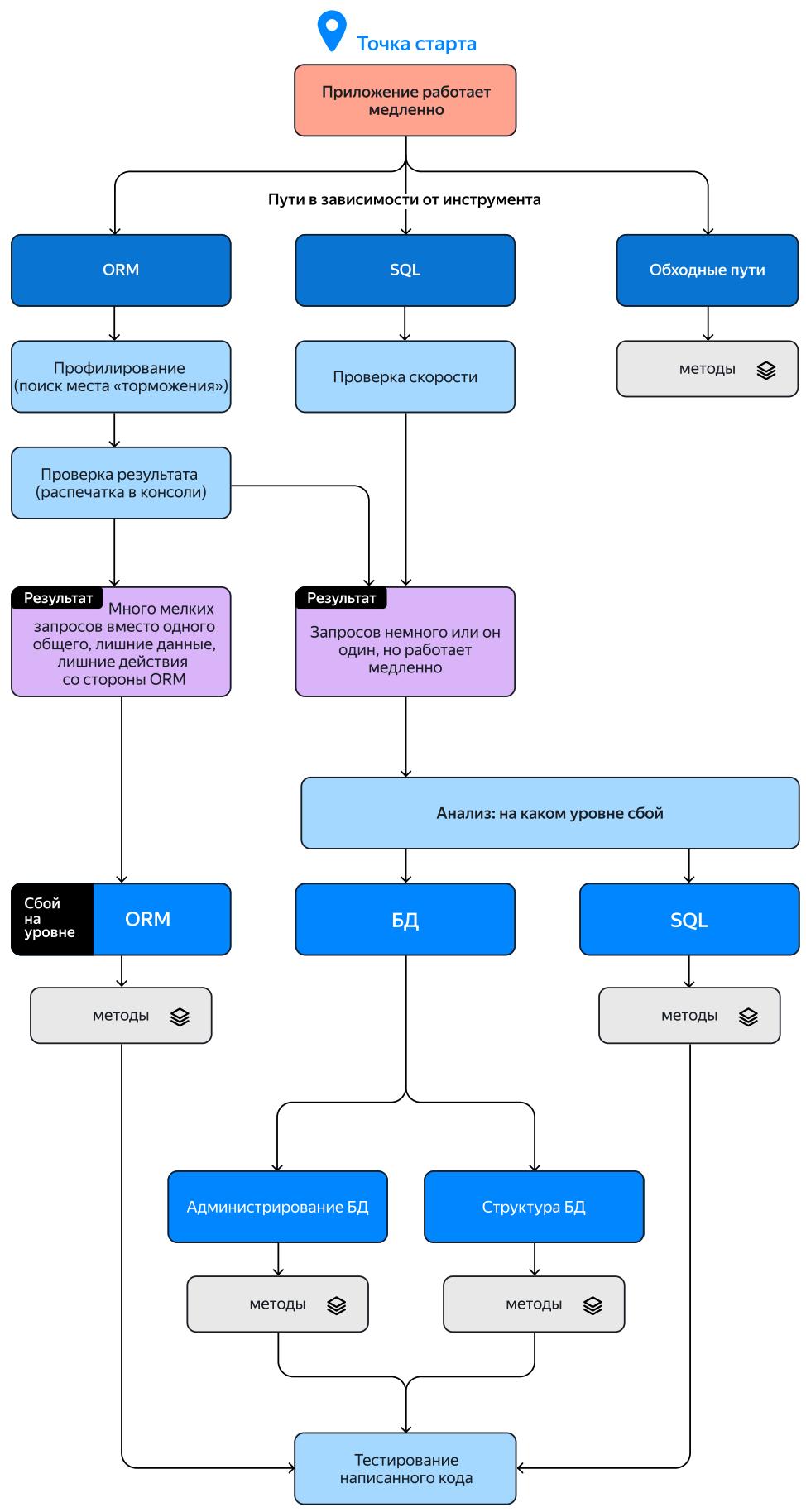
Суть оптимизации — ускорить медленные запросы.

«Медленность» запроса — понятие относительное. В каждом конкретном случае важно детально проанализировать условия, в которых он выполняется, а уже затем определиться — нужна ли оптимизация.

**Шаги разработчика при оптимизации:**

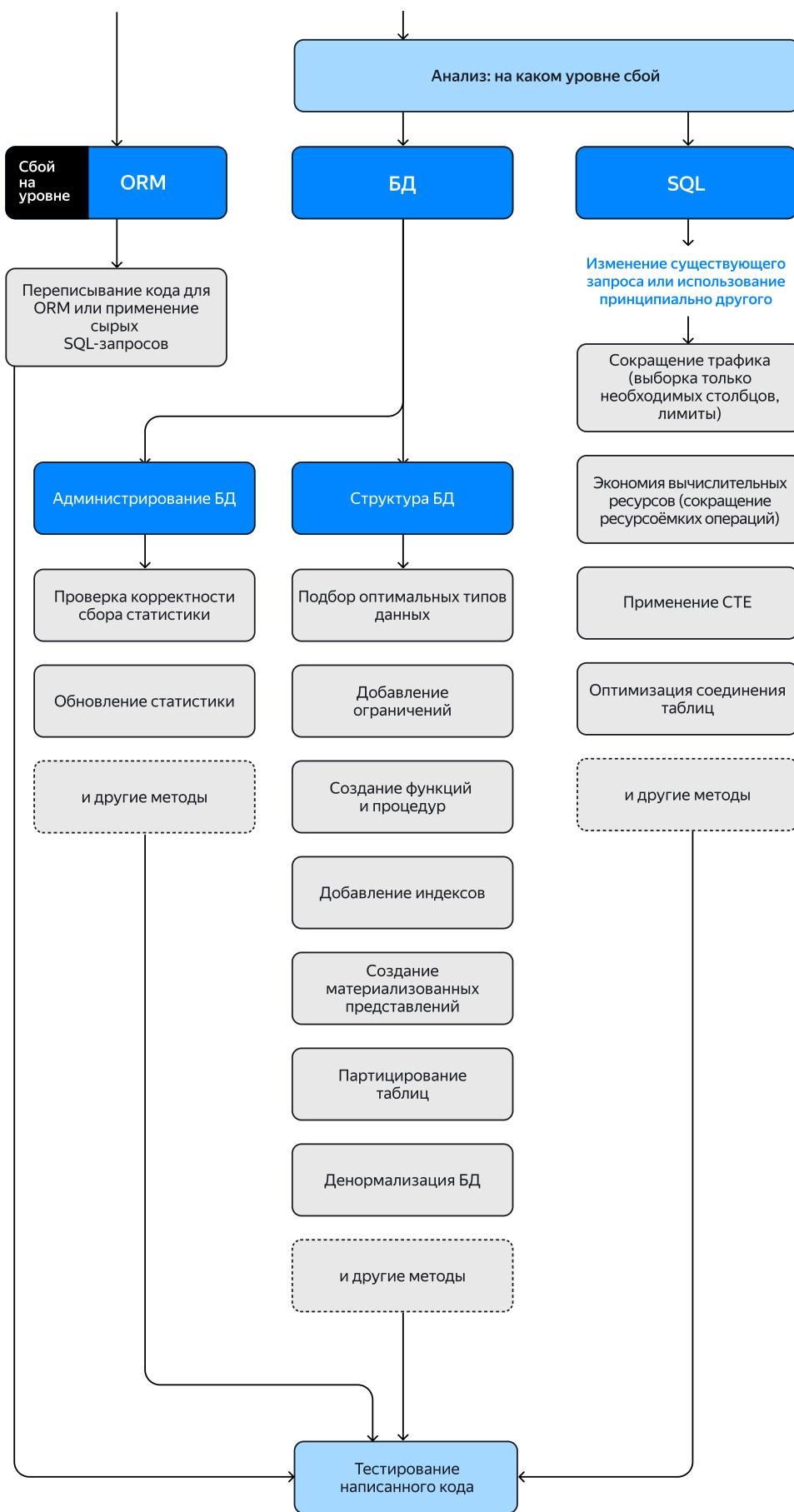
1. Локализует проблему — определяет уровень, на котором она возникла.
2. Выясняет причины, по которым производительность снизилась.
3. Выбирает метод оптимизации на основании полученных данных — подбирает инструмент.

Подробнее — на схеме:



Решение выбрано и внедрено

## Тема 2. Общие подходы к оптимизации



Решение выбрано  
и внедрено

## Тема 2. Общие подходы к оптимизации

### Модуль pg\_stat\_statements

**pg\_stat\_statements** — модуль в составе PostgreSQL. Позволяет получать статистику о запросах, которые обрабатывает сервер. На основании этой статистики можно определить, какие запросы нужно оптимизировать.

#### Основные параметры представления pg\_stat\_statements

Наименование столбца	Тип столбца	Описание
<code>dbid</code>	<code>oid</code>	Идентификатор базы данных, в которой выполнялся запрос.
<code>query</code>	<code>text</code>	Текст SQL-запроса.
<code>calls</code>	<code>bigint</code>	Количество выполнений запроса.
<code>total_exec_time</code>	<code>double precision</code>	Общее время, затраченное на выполнение запроса, в миллисекундах.
<code>min_exec_time</code>	<code>double precision</code>	Минимальное время, затраченное на выполнение запроса, в миллисекундах.
<code>max_exec_time</code>	<code>double precision</code>	Максимальное время, затраченное на выполнение запроса, в миллисекундах.
<code>mean_exec_time</code>	<code>double precision</code>	Среднее время, затраченное на выполнение запроса, в миллисекундах.
<code>rows</code>	<code>bigint</code>	Общее число строк, полученных или затронутых запросом.

### Методы оптимизации на уровне структуры БД

- Выбрать оптимальные типы данных.
- Добавить ограничение **UNIQUE**.
- Создать материализованное представление.
- Изменить структуру базы данных.
- Партицировать (секционировать) таблицы.

## Тема 2. Общие подходы к оптимизации

### Партицирование таблицы

**Партицирование** — это разделение большой таблицы на несколько таблиц меньшего размера (партиции) по какому-то критерию. Таблицы партицируют по определённым критериям (ключам): диапазонам, списку или хешу.

#### Алгоритм партицирования

1. Создать основную таблицу и сообщить БД, что таблица будетパーティцирована. Основная таблица — виртуальная, она не хранится физически и не содержит данных.

Для этого нужно сообщить об этом базе данных с помощью конструкции **PARTITION BY** во время создания таблицы и указать способ партицирования: диапазон, список или хеш, а также по какому полю это делать:

- **PARTITION BY RANGE (поле)** — партицирование по диапазону,
- **PARTITION BY LIST (поле)** — партицирование по списку,
- **PARTITION BY HASH (поле)** — партицирование по хешу.

2. Создать партиции. Партиции — это обычные таблицы, их структура полностью идентична основной таблице. База данных пользуется только её партициями.

Партиции создаются с добавлением служебных слов:

**PARTITION OF** — говорит о том, что это партиция, и в зависимости от метода хеширования указывают критерий:

Метод хеширования	Правило определения	Пример
Диапазон	<b>FOR VALUES FROM</b>	<b>FOR VALUES FROM ('2013-01-01') TO ('2014-01-01')</b> диапазон значений
Список	<b>FOR VALUES IN</b>	<b>FOR VALUES IN (1, 2, 3)</b> <b>FOR VALUES IN ('A', 'B')</b> список значений
Хеш	<b>FOR VALUES WITH</b>	<b>FOR VALUES WITH (MODULUS 3, REMAINDER 0)</b> модуль — <b>MODULUS</b> и остаток от деления — <b>REMAINDER</b>

## Тема 2. Общие подходы к оптимизации

### Планировщик запросов

**Планировщик запросов** — специальный инструмент, с помощью которого перед выполнением каждого запроса PostgreSQL генерирует несколько разных планов, а затем выбирает оптимальный из них — наиболее производительный.

**Поиск оптимального плана запроса** проходит два этапа:

- 1. Анализ запроса.** Планировщик анализирует синтаксис запроса и проверяет правильность его структуры. Ещё он проверяет, есть ли в базе данных таблицы и другие объекты, на которые ссылается запрос, — эта информация есть в системном каталоге.
- 2. Планирование запроса.** В первую очередь планировщик проверяет, есть ли уже скомпилированный план выполнения для этого запроса в кэше планов.
  - Если такой план есть, планировщик использует его и повторно оптимальный план не ищет.
  - Если же скомпилированного плана выполнения в кэше нет или изменена структура базы данных: например, добавлены новые поля или таблицы или изменены типы полей, и это может повлиять на план, планировщик начинает искать оптимальный план запроса.

### Команды и понятия:

- С помощью команды **ANALYZE** собирают актуальную статистику базы данных.
- Команда **EXPLAIN** поможет посмотреть план запроса и оценить его стоимость.
- cost** — стоимость плана запроса. Это предполагаемое количество ресурсов, необходимых для его выполнения.

## Тема 3. Оптимизация на уровне SQL-запросов

### Оптимизация трафика

- Минимизируйте количество столбцов и количество строк в выдаче.

--было

```
SELECT id FROM orders WHERE customer_id = ?
```

--стало

```
SELECT id FROM orders WHERE user_id = ? LIMIT 1;
```

- По возможности выполняйте вычисления на стороне сервера БД.

--было

```
SELECT category, amount, price
FROM orders o
JOIN products p ON o.product_id = p.id
WHERE o.created_at BETWEEN first_of_month AND last_of_month;
```

--стало

```
SELECT
    category,
    COUNT(*) orders_count,
    AVG(amount * price) average_sum
FROM orders o
JOIN products p ON o.product_id = p.id
WHERE o.created_at BETWEEN first_of_month AND last_of_month
GROUP BY category;
```

- Не пересылайте из приложения в БД данные, которые уже в ней хранятся.

--было

```
SELECT id FROM customers WHERE city = 'Смоленск';
SELECT * FROM orders WHERE customer_id IN (список id Смоленских покупателей);
```

--стало

```
SELECT o.*
FROM orders o
JOIN customers c ON o.customer_id = c.id
WHERE customer.city = 'Смоленск';
```

## Тема 3. Оптимизация на уровне SQL-запросов

### Оптимизация вычислений

- Уменьшайте количество используемых функций.
- По возможности стройте запросы без преобразования данных.

--было

```
SELECT COUNT(*) FROM orders
WHERE
    EXTRACT(YEAR FROM created_at) = 2021 AND
    EXTRACT(MONTH FROM created_at) = 5;
```

--стало

```
SELECT COUNT(*) FROM orders
WHERE
    created_at BETWEEN '2021-05-01 00:00:00' AND '2021-05-31 23:59:59';
```

- Большие списки значений в условии **WHERE ... IN** по возможности заменяйте на **JOIN**.

--было

```
SELECT id, name, address, phone, email
FROM customers
WHERE status_id IN (1, 3, 4, 56, 78 ....); -- здесь более 100 вариантов
```

--стало

```
SELECT id, name, address, phone, email
FROM customers
INNER JOIN (VALUES (1), (3), (4) ...) AS statuses (id)
    ON customers.status_id = statuses.id;
```

## Тема 3. Оптимизация на уровне SQL-запросов

- Обсуждайте задачу оптимизации комплексно.

--было

```
SELECT id, name, city, address, phone
FROM customers
WHERE city ILIKE '%city_name%';
```

--стало

```
SELECT id, name AS city_name FROM cities;

SELECT id, name, city, address, phone
FROM customers
WHERE city = 'city_name';
```

## Оптимизация чтения данных

- Избегайте коррелирующих подзапросов, то есть таких подзапросов, которые содержат ссылку на столбцы из включающего его запроса.
- В большинстве случаев соединение предпочтительнее подзапроса.

--было

```
SELECT id, product_id, amount
FROM orders o
WHERE
    o.amount > 2 * (
        SELECT AVG(amount)
        FROM orders
        WHERE o.product_id = orders.product_id
    );
```

--стало

```
SELECT id, amount, o.product_id, avg2
FROM orders o
INNER JOIN (
    SELECT product_id, AVG(amount) * 2 AS avg2
    FROM orders
    GROUP BY product_id
) avg_orders ON
    o.product_id = avg_orders.product_id AND
    o.amount > avg2;
```

## Тема 3. Оптимизация на уровне SQL-запросов

- Страйтесь избегать использования DISTINCT.

```
--было
SELECT DISTINCT p.id, p.name
FROM products p
JOIN orders o ON p.id = o.product_id
WHERE o.created_at BETWEEN '2023-10-01 00:00:00' AND '2023-10-31 23:59:59';
```

```
--стало
SELECT p.id, p.name
FROM products p
WHERE EXISTS (
    -- проверим, есть ли записи о заказах продукта p
    -- за период между '2023-10-01 00:00:00' и '2023-10-31 23:59:59'
    -- так как нам не важны данные заказа, а лишь его наличие
    -- то будем возвращать булево значение true в качестве результата
    SELECT true
    FROM orders o
    WHERE p.id = o.product_id AND
        o.created_at BETWEEN '2023-10-01 00:00:00' AND '2023-10-31 23:59:59'
);
```

- По возможности используйте пагинацию.

## Тема 4. Индексы как способ ускорения запросов

**Индекс** — это объект базы данных, который позволяет быстро находить нужную информацию в большом объёме данных.

**Индекс btree** — самый распространённый тип индексов. В основе работы этого индекса лежит алгоритм бинарного поиска.

**Индекс GIN** — (англ. Generalized INverted index — обратный индекс) при использовании этого индекса индексируются не сами значения, а их элементы. Широко используется для полнотекстового поиска и для поиска вхождений в массивы.

**Индекс GiST** — индекс для работы с данными, которые нельзя сравнивать между собой. Он хорошо работает с геоданными, ip-адресами и диапазонными типами. Также может применяться в полнотекстовом поиске. Этот индекс работает медленнее, чем GIN, но имеет другое преимущество — он строится и обновляется быстрее.

### Создание индекса

Создать индекс можно так:

```
CREATE [UNIQUE] INDEX имя_индекса  
ON имя_таблицы (колонка_1, колонка_2, ...);
```

Если индекс уникальный, укажите это при создании:

```
CREATE UNIQUE INDEX имя_индекса ON имя_таблицы (имя столбца_или_столбцов);
```

При создании составного индекса учитывайте, что его работа зависит от порядка столбцов, указанных в скобках:

- Если индекс создан для двух столбцов — для их комбинации и только первого столбца он будет работать, а для второго — уже нет.
- Если индекс создан для трёх колонок (X, Y, Z) — он сработает для (X, Y, Z), (X, Y) и (X, Z). А вот для (Y, Z) такой индекс уже может не сработать.

### Удаление индекса

Шаблон:

```
DROP INDEX scheme_name.index_name;
```

Здесь **scheme\_name** — название схемы. Его нужно указывать, поскольку по умолчанию индекс не находится в схеме.

## Тема 4. Индексы как способ ускорения запросов

### Другие типы индексов

Индекс	Описание	Как создать
Частичный	Создаётся на подмножестве строк таблицы, которое соответствует определённому условию	При помощи ключевого слова <code>WHERE</code> и условия. Эта конструкция помещается в конец команды <code>CREATE INDEX</code>
Покрывающий	Содержит не только индексируемый столбец, но и дополнительные столбцы, которые не обрабатываются механизмом индекса	При помощи ключевого слова <code>INCLUDE</code>
Функциональный	Строится не по столбцу, а по выражению	При создании индекса указать функцию, которая применяется к столбцу
Функциональный для поиска по шаблону	Применяется для запросов, в которых осуществляется поиск по части поля	Добавить к колонке ключевое слово: - <code>text_pattern_ops</code> для данных типа <code>text</code> , - <code>varchar_pattern_ops</code> для типа <code>varchar</code> , - <code>bpchar_pattern_ops</code> для <code>char</code> .

### Системные представления для анализа индексов

Позволяют посмотреть:

- `pg_stat_statements` — скорость выполнения запросов.
- `pg_stat_all_tables` — обращения ко всем таблицам БД.
- `pg_stat_user_tables` — обращения к пользовательским таблицам.
- `pg_stat_user_indexes` — как используются пользовательские индексы.

## Тема 5. Чтение и анализ плана запросов

План запроса можно представить в виде дерева, которое демонстрирует последовательность шагов выполнения запроса.

- Каждый шаг, или «узел», — это одна операция. Узел в первой строке плана называются «корневым». Все остальные узлы, кроме корневого, помечают символом `->`.
- Каждый узел-операция передаёт результат своей работы вышестоящему узлу, поэтому план читают снизу вверх. На самом верху (на вершине дерева) — операции, которые выполняются в последнюю очередь.
- Если несколько узлов находятся на одном уровне — значит, эти операции не зависят друг от друга и могут выполняться параллельно.

## Получение плана запроса

Ожидаемый план запроса получают командой `EXPLAIN`. При этом сам запрос не выполняется.

```
EXPLAIN SELECT COUNT(*) FROM tools_shop.users;
```

Чтобы выполнить запрос и получить фактический план с реальными затратами, используют команду `EXPLAIN` с параметром `ANALYZE`:

```
EXPLAIN ANALYZE SELECT COUNT(*) FROM tools_shop.users;
```

## Операции получения данных

Эти операции используются для чтения из таблиц и индексов.

- `Seq Scan` — последовательное чтение.
- `Index Only Scan` — чтение только индекса.
- `Index Scan` — поиск нужной страницы в индексе и переход к ней.

## Тема 5. Чтение и анализ плана запросов

### Операции обработки данных

Такие операции используются для сортировки, лимита, агрегации и т. д.

#### Limit

Limit ограничивает количество возвращаемых строк. Её используют, если в запросе есть ключевые слова LIMIT или OFFSET.

Когда операция Limit получает от дочерней операции нужное количество данных, она эту операцию останавливает. При этом, если в запросе есть OFFSET, то в операцию Limit обрабатываются и передадутся:

- все строки, на которые смещается выборка,
- плюс строки, которые нужно отобразить.

И уже в узле Limit лишние строки отсеются.

#### Sort

Sort сортирует данные по указанным полям. Её используют:

- В планах запросов с ORDER BY.
- Если в плане есть операции, которым нужна предварительная сортировка.

### Ещё операции обработки данных:

- WindowAgg применяет оконную функцию.
- GroupAggregate агрегирует данные, отсортированные по ключу агрегации.
- HashAggregate агрегирует строки, используя хеш-таблицу. Эту операцию применяют, когда нужно агрегировать большой объём неотсортированных данных.

## Тема 5. Чтение и анализ плана запросов

### Операции соединения таблиц

Такие операции используются для выполнения **JOIN**.

В плане запроса эти соединения отражены одной из трёх операций: **Nested Loop**, **Hash Join**, **Merge Join**. Эти операции называют физическим соединением таблиц.

Операция соединения	Как соединяет таблицы	Для чего подходит
<b>Nested Loop</b>	вложенными циклами	- соединения небольших объёмов данных - соединения строк не только по равенству атрибутов, но и по другим условиям: больше, меньше, не равно
<b>Hash Join</b>	с помощью хеш-таблицы	- быстрого соединения неотсортированных наборов - соединения только по равенству атрибутов
<b>Merge Join</b>	слиянием	отсортированных наборов данных любых размеров

### Анализ плана запроса

Анализируя план запроса, обратите внимание на пункты:

- **Статистика.** Параметр **rows** в плане запроса поможет проверить актуальность статистики для таблиц, которые используются в запросе.
- **Стоимость выполнения.** У каждого узла в плане запроса есть стоимость — **cost**. Отдельные узлы могут значительно влиять на общую стоимость плана запроса.
- **Метод чтения таблиц и применение индексов.** Таблицы могут читаться полностью и последовательно операцией **Seq Scan** или сканироваться с применением индексов операциями **Index Scan**, **Index Only Scan** и **Bitmap Index Scan**. Использование индексов может существенно повысить производительность запроса.
- **Метод соединения таблиц.** Таблицы могут соединяться медленно операцией **Nested Loop** или быстро операциями **Hash Join** или **Merge Join**. Переписав запрос, можно изменить способ соединения и повысить производительность.

Анализ только одного параметра может быть недостаточным. Чтобы сделать корректные выводы, оценивайте эти параметры только комплексно, в совокупности.