1. Выполнение EXPLAIN ANALYZE

**Изображение выглядит как текст, чек, Шрифт, белый

Автоматически созданное описание**

**cost** – стоимость конкретной операции (напротив которой написан), в условных единицах (считаются через **(disk pages read \* seq\_page\_cost) + (rows scanned \* cpu\_tuple\_cost)**.)

1 число – время, которое пройдет до получения из этой операции первой строчки

2 число – время, которое пройдет до получения последней строчки

**rows** – ожидаемое количество строк в результате

**width –** ожидаемый средний размер строки в байтах

**loops –** кол-во выполнений операции

варьируется в зависимости от:

вложенных циклов,

**использования индексов** (несколько проходов по индексным страницам),

**реализация алгоритмов соединений** (**nested loop join** – внутренний цикл повторяется для каждой строки внешнего цикла, **hash join** – создание хэша и проверка каждой строки, **merge join** – проход по отсортированным строкам),

**кэширование** и повторное использование результатов (в некоторых случаев пскл может кэшировать результаты подзапросов/промежуточные результаты, и повторно использовать их в частях основного запроса)

**параллелизм** (таблица разбита на партии, каждая партия обрабатывается параллельно => каждая из операций имеет свои циклы выполнения, которые будут суммированы в общий результат)

**нулевой** – если предыдущий узел отфильтровал уже все, следовательно, нынешнему узлу не требуется фильтровать ничего

Порядок выполнения операций – по уровням. То есть выбирается то, что находится на одном уровне и последовательно выполняется.

2. Чем отличаются Seq Scan, Index Scan, Index Only Scan?

**Sequential Scan (Seq Scan)**

**Seq Scan** выполняет последовательное сканирование всей таблицы. PostgreSQL читает каждую строку таблицы одну за другой и проверяет, соответствует ли она условию запроса.

**Когда используется**:

* Когда таблица небольшая, и стоимость последовательного сканирования незначительна.
* Когда условия запроса не могут эффективно использовать индекс.
* Когда доля возвращаемых строк велика (например, более 5-10% от общего числа строк), последовательное сканирование может быть быстрее, чем индексное сканирование.

**Index Scan**

**Index Scan** использует индекс для нахождения строк, соответствующих условию запроса. После нахождения строки в индексе PostgreSQL обращается к соответствующей строке в таблице для получения всех необходимых данных (если индекс не покрывает все столбцы).

**Когда используется**:

* Когда запросы содержат условия, по которым существует индекс.
* Когда требуется выборка небольшого числа строк.

### Когда индекс помогает быстро находить необходимые строки, особенно при использовании условий поиска или сортировки.

### **Index Only Scan**

**Описание**:

**Index Only Scan** также использует индекс для нахождения строк, но отличается тем, что все необходимые данные находятся в самом индексе. Это возможно, если индекс покрывает все запрашиваемые столбцы, и PostgreSQL не нужно обращаться к таблице для получения дополнительных данных.

**Когда используется**:

* Когда запросы запрашивают только те столбцы, которые включены в индекс.
* Когда индекс содержит всю необходимую информацию для запроса.

3. Чем отличается Index Cond от Filter?

Filter и Index Cond являются важными элементами, описывающими, как и где применяются условия поиска и фильтрации данных.

### Index Cond

**Описание**:

* **Index Cond** (Index Condition) указывает условия, которые используются непосредственно для поиска строк в индексе. Эти условия помогают PostgreSQL быстро находить нужные строки, используя структуру индекса.

**Когда используется**:

* Когда условия в запросе могут быть удовлетворены с помощью индекса.
* Например, условия равенства (=), диапазона (BETWEEN, <, >) и другие, которые могут эффективно использовать индекс.

### Filter

**Описание**:

* **Filter** указывает условия, которые применяются к строкам после их извлечения из таблицы или индекса. Эти условия фильтруют данные, отбирая только те строки, которые удовлетворяют условию.

**Когда используется**:

* Когда условия не могут быть использованы для индексного поиска.
* Когда дополнительные условия должны быть применены после извлечения данных.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

4. nested loop join, hash join и merge join:

Nested Loop Join выполняет соединение таблиц путем вложенного цикла. Для каждой строки из первой таблицы (внешняя таблица) он перебирает все строки из второй таблицы (внутренняя таблица) и проверяет условие соединения.

**Принцип работы**:

1. Для каждой строки из внешней таблицы:
   * Для каждой строки из внутренней таблицы:
     + Проверить условие соединения.
     + Если условие выполняется, добавить комбинацию строк в результат.

**Когда используется**:

* Когда одна из таблиц очень маленькая.
* Когда условие соединения не может быть эффективно использовано с другими методами соединения.
* Когда количество строк в таблицах относительно небольшое.

Hash Join использует хеш-таблицу для выполнения соединения. Он создаёт хеш-таблицу из одной таблицы (обычно из меньшей таблицы), а затем пробегает по другой таблице, сравнивая строки с хеш-таблицей.

**Принцип работы**:

1. Создать хеш-таблицу:
   * Выбрать меньшую таблицу (внутреннюю таблицу).
   * Создать хеш-таблицу на основе значений соединяющего столбца.
2. Пробежаться по внешней таблице:
   * Для каждой строки из внешней таблицы:
     + Найти соответствующие строки в хеш-таблице.
     + Если условие соединения выполняется, добавить комбинацию строк в результат.

**Когда используется**:

* Когда одна из таблиц относительно мала.
* Когда обе таблицы не отсортированы по условию соединения.
* Когда необходимо быстрое соединение для больших наборов данных.

Merge Join требует, чтобы обе таблицы были отсортированы по столбцу соединения. Он проходит по обеим таблицам, сравнивая строки, и если находит совпадение, добавляет их в результат.

**Принцип работы**:

1. Отсортировать обе таблицы по столбцу соединения (если они ещё не отсортированы).
2. Пройти по обеим таблицам параллельно:
   * Сравнивать текущие строки из обеих таблиц.
   * Если строки совпадают по условию соединения, добавить комбинацию строк в результат.
   * Если значение из одной таблицы меньше, перейти к следующей строке этой таблицы.

**Когда используется**:

* Когда обе таблицы отсортированы по условию соединения или легко отсортировать.
* Когда объем данных велик, и другие методы соединения менее эффективны.
* Для диапазонных условий соединения.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

**Nested Loop Join** прост и эффективен для небольших наборов данных, **Hash Join** хорош для больших неупорядоченных данных, а **Merge Join** эффективен для предварительно отсортированных данных.

5. Типы индексов

PostgreSQL provides several index types: B-tree, Hash, GiST, SP-GiST, GIN, BRIN, and the extension bloom (bloom provides an index access method based on Bloom filters. A Bloom filter is a space-efficient data structure that is used to test whether an element is a member of a set. In the case of an index access method, it allows fast exclusion of non-matching tuples via signatures whose size is determined at index creation.).

**B-TREE**

B-trees can handle equality and range queries on data that can be sorted into some ordering. In particular, the PostgreSQL query planner will consider using a B-tree index whenever an indexed column is involved in a comparison using one of these operators:

**< <= = >= >**

Constructs equivalent to combinations of these operators, such as BETWEEN and IN, can also be implemented with a B-tree index search. Also, an IS NULL or IS NOT NULL condition on an index column can be used with a B-tree index.

B-tree индексы могут быть использованы для сортировки, ограничений уникальности и поиска по диапазону значений.

**HASH**

Hash indexes store a 32-bit hash code derived from the value of the indexed column. Hence, such indexes can only handle simple equality comparisons. The query planner will consider using a hash index whenever an indexed column is involved in a comparison using the equal operator:

**=**

**GiST (Generalized Search Tree)**

GiST-индексы являются обобщенными и многоцелевыми, предназначены для работы со сложными типами данных, такими как геометрические объекты, текст и массивы. Они позволяют быстро выполнять поиск по пространственным, текстовым и иерархическим данным.

**<< &< &> >> <<| &<| |&> |>> @> <@ ~= &&**