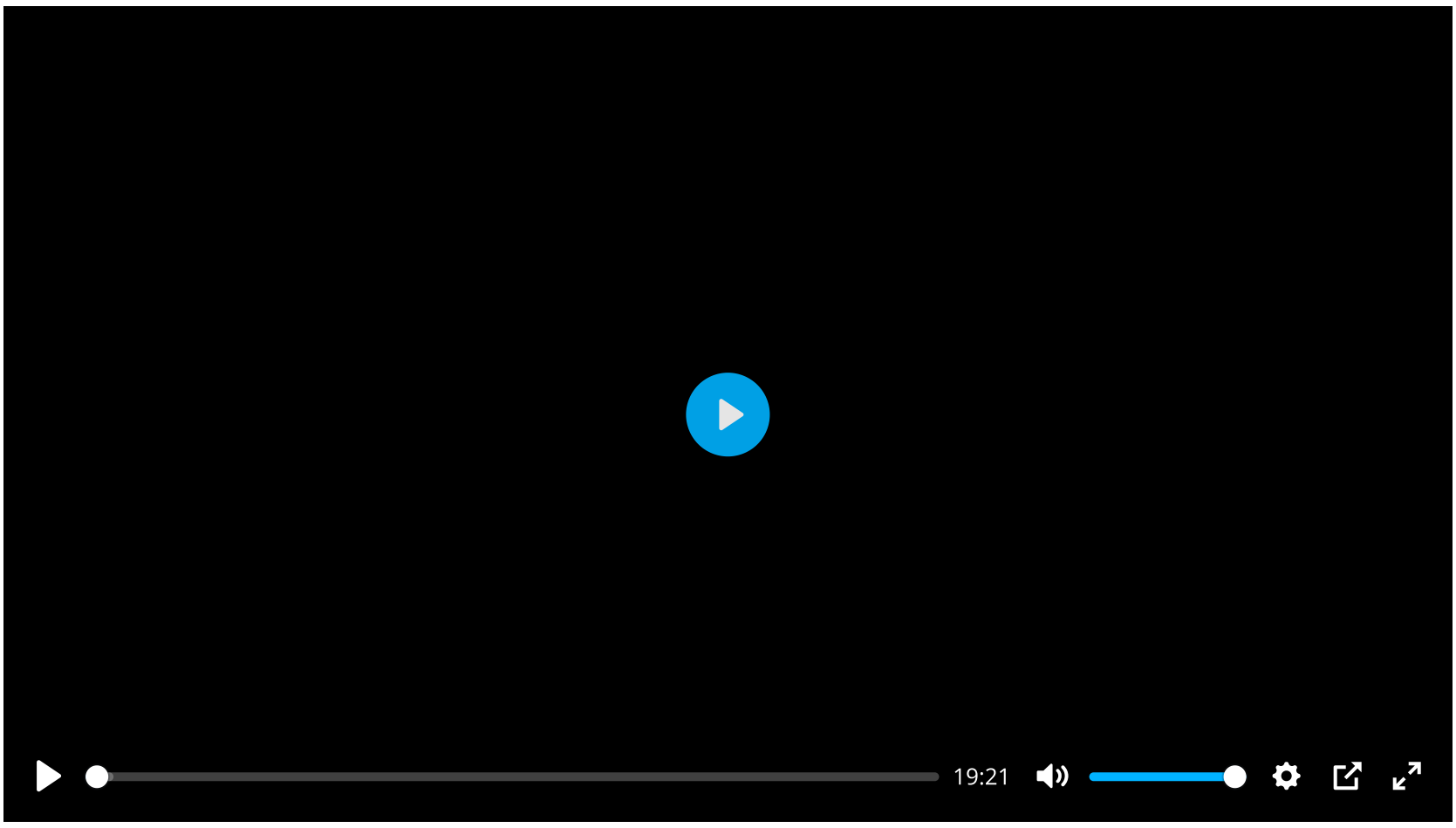
5.4 База данных глазами Data Engineer'a



Текстовая расшифровка видео:

ИНДЕКСЫ И EXPLAIN

План:

- Я сделал key-value store;
- Log-Structured Merge Tree;
- Индекс;
- Как мы ищем;
- Как исполняется запрос: AST;
- Трансформация и планирование;
- Сортировка и ресурсы.

Я сделал key-value store

Рассмотрим базовый пример:

```
#!/usr/bin/env bash

db_set() {
  echo "$1,$2" >> database
}

db_get() {
  grep "^$1," database | sed -e "s/^$1,//" | tail -n 1
}
```

Это база – ключ-значение, которая использует плоский текстовый файл. Когда мы выставляем к ключу-значения, мы записываем в конец файла пару через запятую (ключ, значение). Когда хотим его извлечь, мы регуляркой достаем из файла последнюю строчку, у которой первое поле этого ключа совпадает с

тем, что ищем. Достаем значение и возвращаем результат для этого ключа и значения.

Что не так с данным примером?

К чему можно придраться:

- Нет компактизации;
- Каждый раз нужно просматривать файл целиком и т.д.

Log-Structured Merge Tree

Существует БД:

- оптимизированная на запись;
- оптимизированная на чтение.

Один из механизмов оптимизации записи называется «**Log-Structured Merge Tree**».

Когда БД работает с диском (особенно, если это магнитный диск), самым быстрым механизмом записи будет последовательная запись (блоки скидываем на диск по порядку).

Представим, что у нас есть данные, упорядоченные по определенному ключу. Мы пишем их в базу и хотим, чтобы база максимально быстро их читала и записывала на диск. Эта база может поддерживать в памяти специальную структуру (в нашем случае – дерево).

За счет того, что это дерево мы достигаем сразу несколько целей:

- Записываем данные в дерево, вставляя в уже рассортированной последовательности (дерево поддерживает порядок).
- Поскольку дерево поддерживает порядок, мы можем его обойти и получить данные в отсортированном виде.

Таким образом, вставки происходят довольно быстро, при этом в отсортированной последовательности, также происходит быстрое сохранение на диск.

Популярные базы:

- Casandra;
- SCYLLA;
- RocksDB.

LSM-Tree paper

Google BigTable paper

Индекс

Вопрос: что из себя представляет индекс?

Ответ: у нас есть какие-то данные, и мы хотим иметь возможность находить отдельные записи по каким-то параметрам, при этом делать это не последовательным просмотром (линейно), а быстрым способом.

Типичный способ – поднять словарь и взаимодействовать с ним.

Требования к процедуре поиска:

- Результат запроса с индексом и без индекса должен быть идентичным;
- Индексы ускоряют поиск, сортировку, группировку и JOIN’ы;
- Нужно много места и затрат на перестройку;

- Индексы – это забота в первую очередь программиста, а уже потом DBA.

Многие из вас знают, что **Elasticsearch** – полнотекстовый поисковой движок, позволяющий грузить документы и искать по ним что-либо. Его часто используют в качестве back-end поиска на сайтах. Внутри он представляет из себя пачку разных индексов.

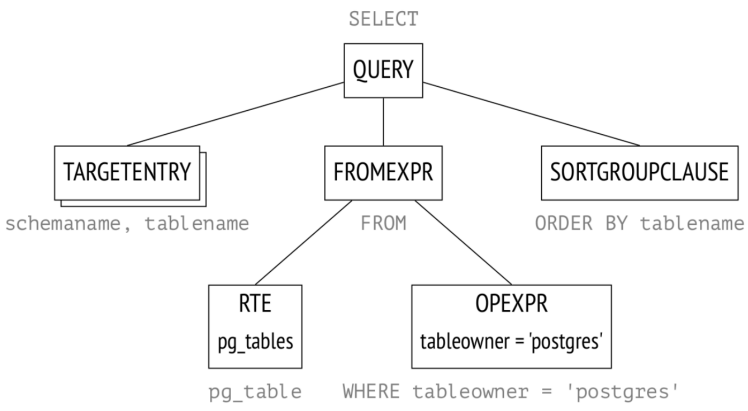
Как мы ищем

Способы поиска:

- **Sequential Scan** по предикату;
- **Ordered Index** — бинарный поиск в индексе, но случайное чтение таблицы (если данные отсортированы по какому-то полю, а запрос от нас поступает по другому полю, то имеет смысл создать сортированную структуру данных по нужному нам полю с ссылками на ряды в табличке. Мы потеряем в последовательном чтении с диска, но приобретем возможность быстрого извлечения данных);
- **Hash Index** — «запустим рядом Redis»;
- **B-Tree Index** — удобно искать, чаще всего по умолчанию;
- **Clustered Index** — физическая сортировка таблицы на диске.

Как исполняется запрос: AST

Когда мы запускаем какой-то запрос по индексированным/неиндексированным данным, происходит то же, что и при выполнении кода на любом языке программирования: строится абстрактное синтаксическое дерево с разными нодами:



Трансформация и планирование

Оптимизатор будет смотреть на построенное дерево и будет предполагать, что можно перенести или выкинуть, чтобы запрос выполнялся наиболее эффективно.

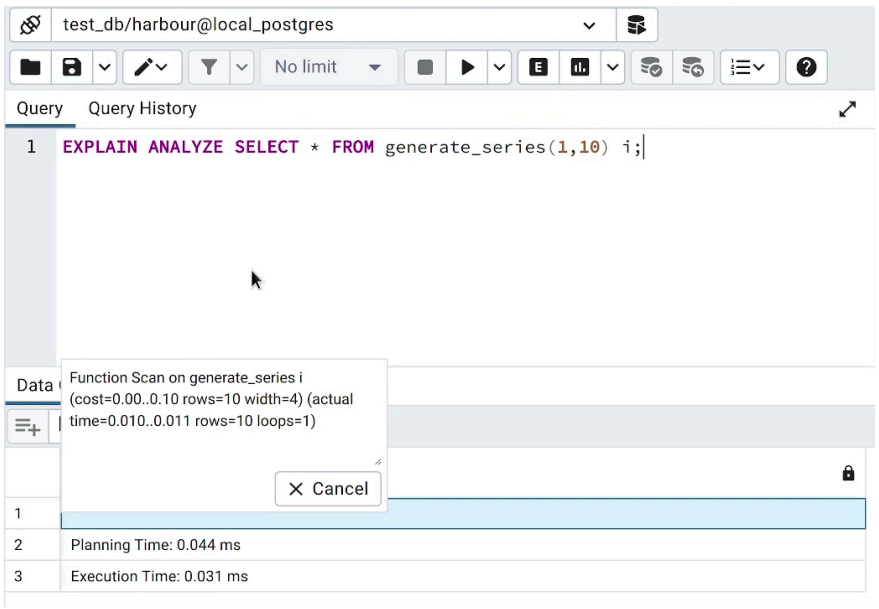
- **Row-level security** работает на стадии трансформации;
- PostgreSQL использует **cost-based query optimizer** + «метод ветвей и границ»;
- Apache Spark использует SQL как промежуточное представление, поверх него запускает свой собственный **Catalyst Optimizer**.

Для того, чтобы понять, как база будет пытаться выполнить запрос, можно использовать «Explain».

Выполним Explain analyze с подобным запросом:

```
EXPLAIN ANALYZE SELECT * FROM generate_series(1,10) i;
```

Мы генерируем последовательность данных и проходимся по ней. Видим следующее:



Мы линейно прошлись по генерированным результатам и вывели все, что получилось.

В Postgres есть два варианта вызова Explain:

- Explain (даем query оптимизатору, он его оптимизирует и возвращает приблизительный пример выполнения);
- Explain analyze (физически выполняет запрос, при этом мы получаем более полный отчет).

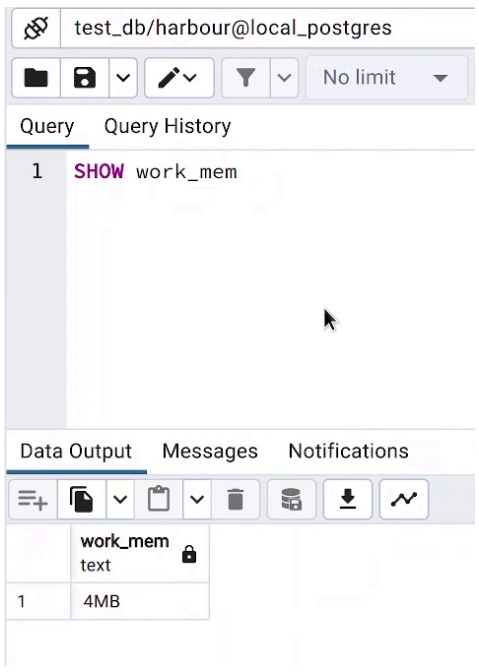
Сортировка и ресурс

Так, например, мы генерируем последовательность и сортируем ее по рандомным параметрам.

Нюанс: когда мы делаем группировку или упорядочивание, в идеале PostgreSQL старается делать эти операции в памяти. Все данные, которые мы сортируем, можем поместить в памяти. Соответственно нам ничего не мешает применить стандартный алгоритм сортировки. PostgreSQL выполняет quicksort. Если же данные не помещаются в память, получается более сложная ситуация, так как необходимо выполнить Out Of Course-сортировку, то есть нужно часть данных выгрузить на диск и отсортировать.

Объем памяти, использующийся при подобных вычислениях, называется «Work_mem».

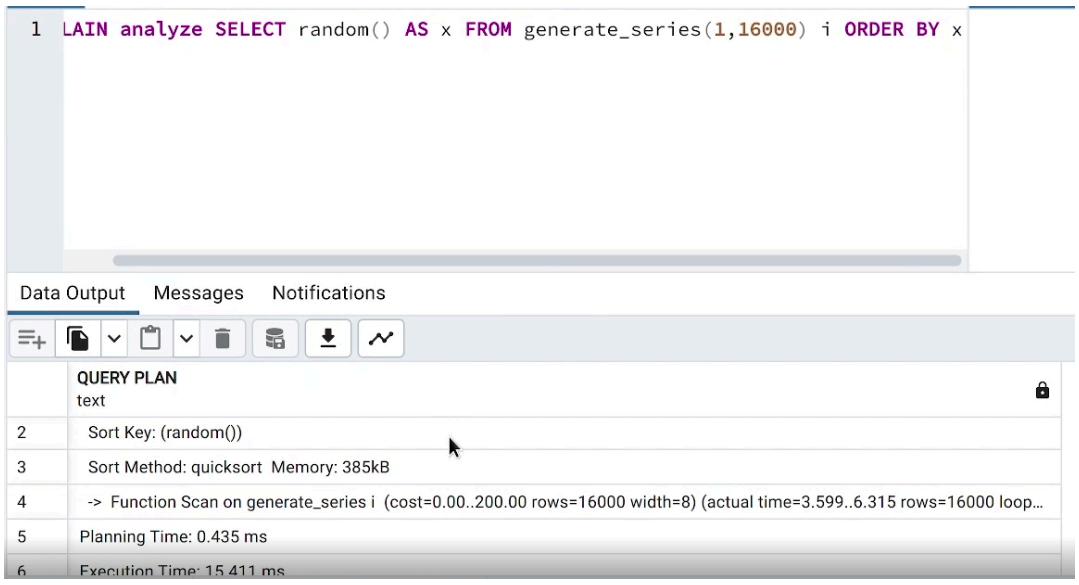
Значение по умолчанию у Work_mem:



Выполним подобный запрос:

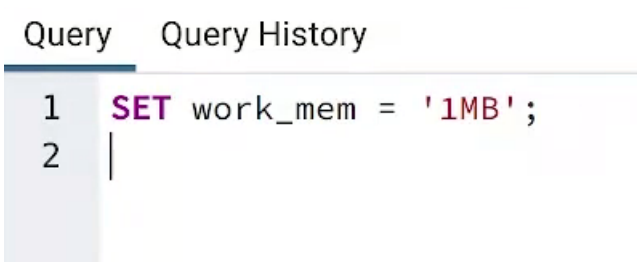
```
EXPLAIN analyze SELECT random() AS x FROM generate_series(1,16000) i ORDER BY x
```

Получится следующее:



Все работает.

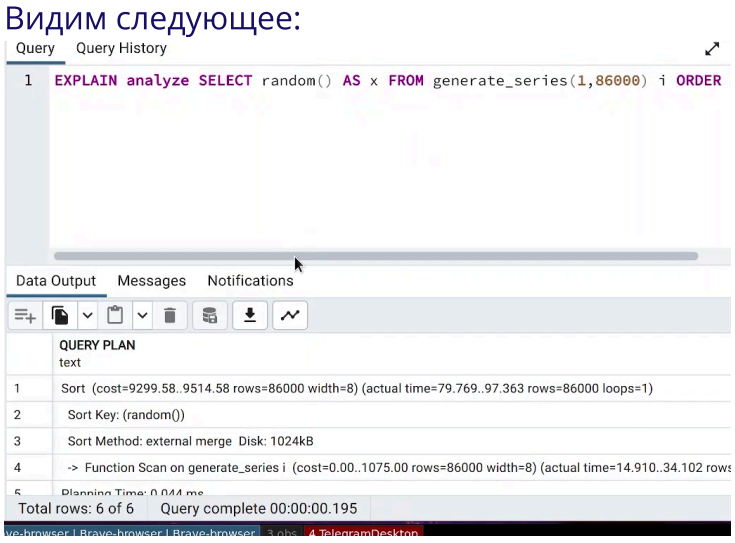
Проведем эксперимент и подправим значение Work_mem так, чтобы данные специально не поместились:



Сгенерируем последовательность (она достаточно большая и может не влезть в Work_mem, отсортироваться у нее тоже так просто не получится):

```
EXPLAIN analyze SELECT random() AS x FROM generate_series(1,86000) i ORDER BY x
```

Видим следующее:



Поскольку в Work_mem оно не поместилось, пришлось использовать диск. Postgres'у пришлось использовать алгоритм «External merge». Заметим, что данные физически выгружались.

Ссылка на почитать

Как вам урок?

Изучил, далее

Слёрм ©

+7 (495) 248-05-80

Лицензия №ДЛ-1368 от 22.08.2019

Политика конфиденциальности

Публичная оферта