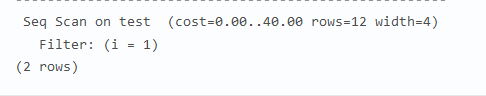
**ПЛАН ЗАПРОСА**

Получить план запроса можно с помощью команды **explain** после которой следует сам запрос. Пример вывода



Вначале указывается каким образом postgres будет искать данные. Затем указывается таблица.

**Cost –** стоимость операции, это число оценивает временные и ресурсные затраты на выполнение операции.

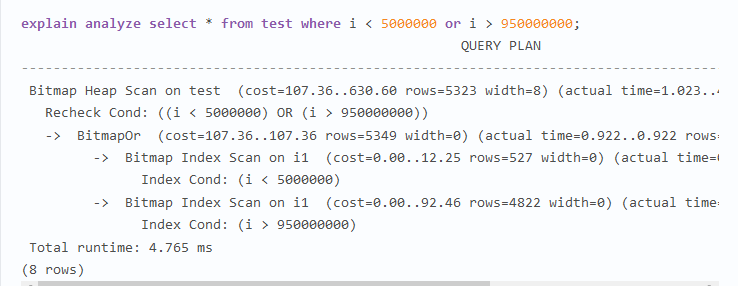
**Rows –** приблизительное количество строк, которое эта операция способна вернуть.

**Width –** приблизительно столько байт содержится в одной строке, возвращенной в рамках данной операции.

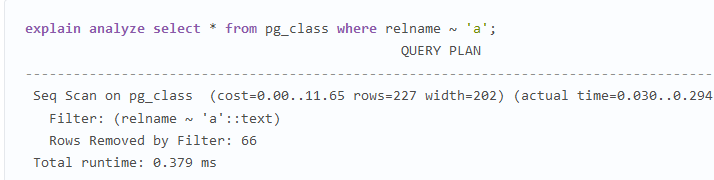
Первое число указывает издержки, необходимые для получения первого элемента операции, второе – на получение всех строк.

Можно вывести также фактическое время, которое потребовалось на выполнение операции, используя **explain analyze.**

Виды операций в плане запроса:

* **Seq Scan on table\_name** – последовательное сканирование.
* **Index Scan using index\_name on table –** поиск по индексу. Postgres открывает индекс; в индексе находит, где в таблице находятся строки, соответствующие условию; открывает таблицу и получает строки, указанные индексом.  
  Сканирование по индексу также может использоваться при сортировке по проиндексированному полю.
* **Index Scan Backward –** сканирование по индексу в порядке убывания. Выполняется например в таком случае:  
  
* **Index Only Scan –** Выполняется когда мы выбираем колонки только из индекса, и нет необходимости лезть в саму таблицу. Данные будут возвращаться прямо из индекса.
* **Limit** – соответствует sql команде LIMIT  
  
* **Bitmap Heap Scan** в комбинации с **Bitmap Index Scan –** Вначале Bitmap Index Scan создает битовую карту, где каждой странице из таблицы будет соответствовать 1 бит. Затем он установит биты в единички, если на этой странице может находится строка, которую надо вернуть. После того как будут найдены все страницы, Bitmap Heap Scan выполняет последовательный поиск в них. Несколько условий создают несколько Bitman Index Scan, которые затем объединяются с помощью **Bitmap Or, And или Not.**

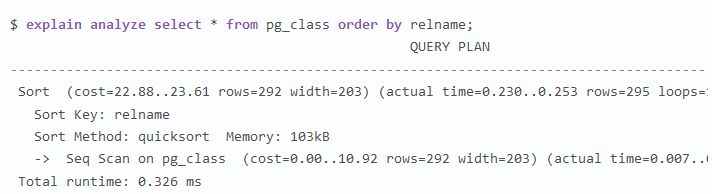
В операциях с условием WHERE план запроса содержит строку **Filter**, в которой указано, по каким критериям осуществлялась фильтрация и сколько строк было отброшено.



В операциях с ORDER BY план запроса содержит операцию **Sort**. В качестве доп информации указаны поле для сортировки и ее тип.

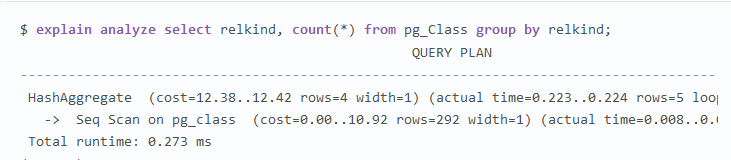
Если тип сортировки **Memory,** используется оперативная память.

Ели тип сортировки **Disk –** postgres использует временные файлы.



**HashAggregate –** эта операция в основном применяется при использовании GROUP BY и каких-нибудь агрегатных функций.

Перед выполнением этой операции сканируются все строки. Для каждой строки находит «ключ» по которому происходит группировка. Затем в хэше (ассоциативном массиве) добавляет выбранную строку в нужную корзину. После обработки всех строк, хэш сканируется и выполняется расчет агрегатных функций.

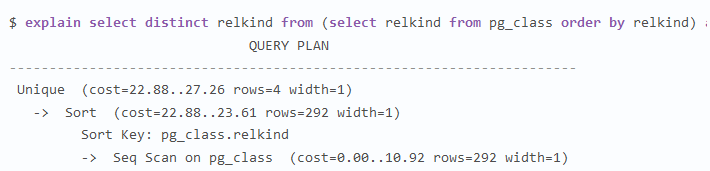


**HashJoin –** используется при объединении таблиц с помощью join.

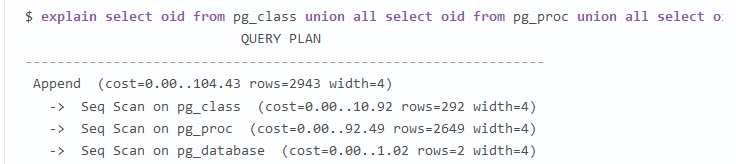
Сначала HashJoin вызывает субоперацию **Hash,** которая в свою очередь вызывает обход таблицы. Hash создает ассоциативный массив, ключом которого является столбец по которому происходит объединение.

Потом Hash Join запускает вторую субоперацию и для каждой строки из нее ищет совпадение в хэше. Если совпадений нет, строка игнорируется. Если есть – то формируются совпадающие строки.

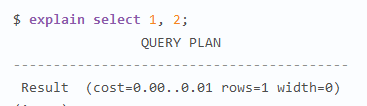
**Unique –** отбор уникальных строк, используется при вызове SQL оператора DISTINCT, но только когда строки отсортированы. Она сравнивает текущее значение с предыдущим, и если они одинаковые, отбрасывает текущее.



**Append –** запускает множество субопераций, и возвращает все возвращенные ими строки в виде общего результата. Используется в запросах с UNION и UNION ALL



**Result** – появляется в очень простых запросах, как правило, когда выбирается постоянное значение.



Планировщик PosthreSQL выбирает операции основываясь на статистике.

Например представим такой запрос:



Если у всех строк в таблице одинаковое значение some\_value, тогда применение к столбцу индекса не имеет смысла.

С другой стороны, если значения в столбце уникальны (или почти уникальны), использование индекса – отличная идея.