USAGE OF INDEXES AND SEQUENTIAL SEARCH BASED ON QUERIES

This report describes how PostgreSQL chooses to use index or sequential search depending on the query it receives.

This report analyses which type of search is more appropriate and which one is slow and bad.

What is an index?

Index is a data structure, created alongside a table, which is used to speed up searches or sorts.
The index has to be kept up to date, so every insert/update/delete operation gets a bit more expensive.

How does PostgreSQL use index, when it chooses to?
Let's assume we have some table, with column ABC_XYZ (integer, not unique), and index on it.
If PostgreSQL will choose to use it, let's assume the query is:

```
SELECT * FROM table WHERE ABC_XYZ = 123
```

It will:

- find record in index
- based on information from index, it will find appropriate page and record in table (heap data)

Finding record in index is relatively complicated (although usually fast) operation, which can easily involve multiple random page reads.

At disk level, there are two different kinds of access:

- sequential next block of data is read just after previous, so no disk head movement is needed
- random next block is read from random place on disk, which requires head movement, and thus is slower

So, in best case, index scan requires (from disk):

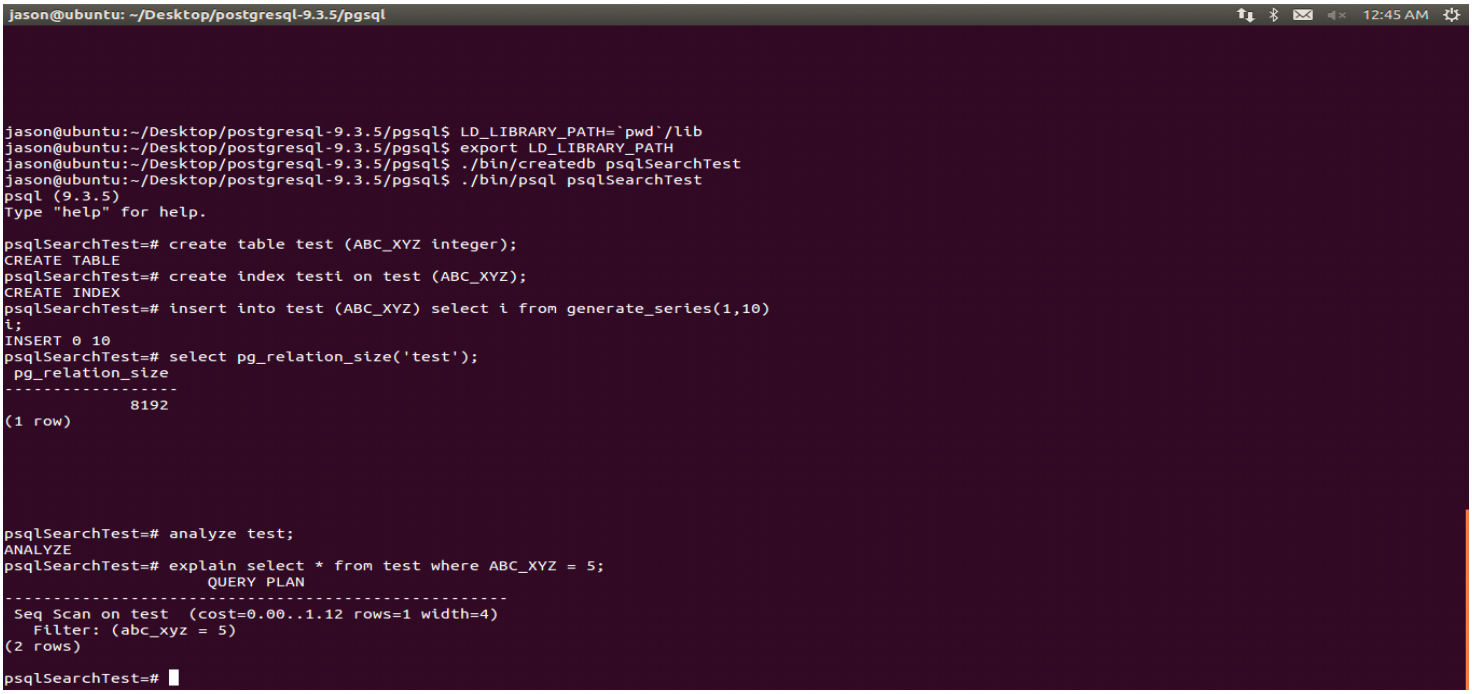
- seek to where index is
- fetch page from index
- seek to where table is
- fetch page from table

On the other hand, best case for sequential scan is simpler:

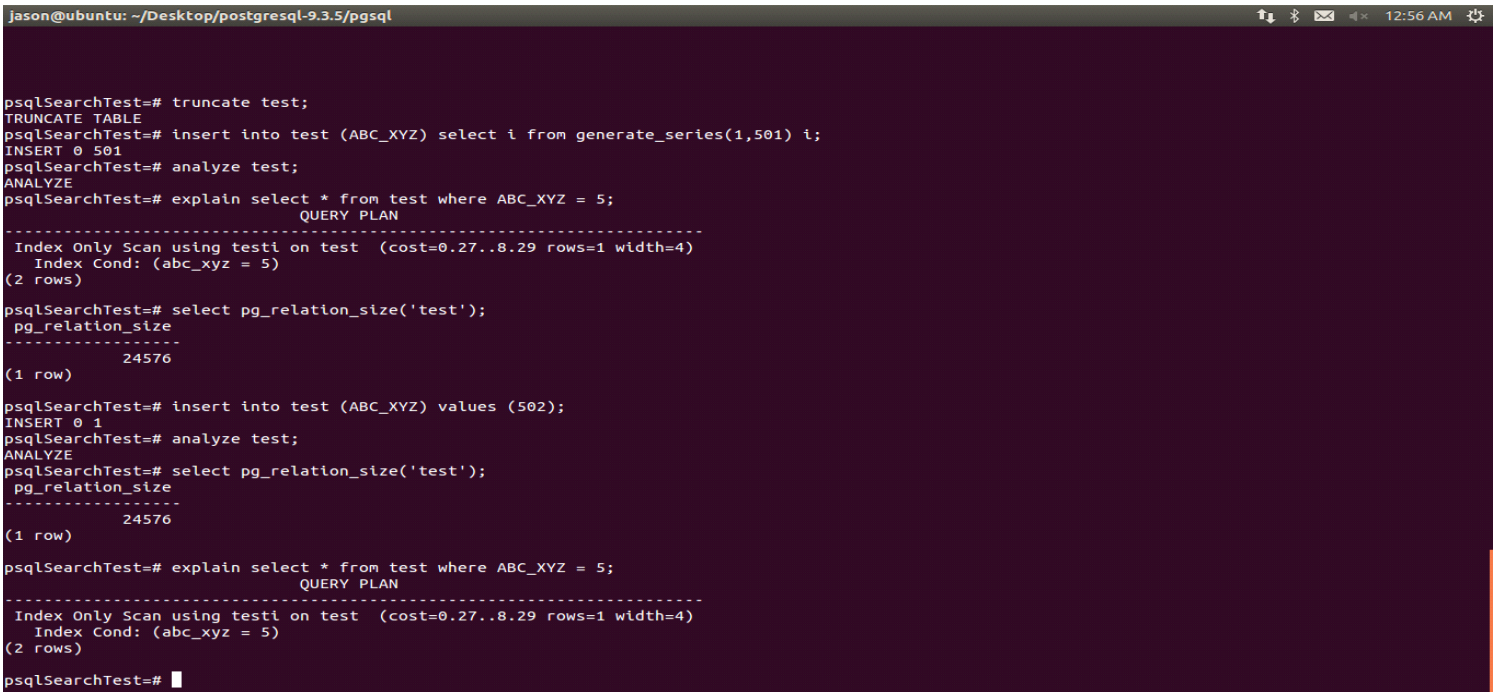
- seek to where table is
- fetch page from table

SITUATION 1

This can mean that if table is smaller than 2 pages (1 page is 8192 bytes), PostgreSQL shouldn't use index at all! In reality it's probably a bit more complicated, so the easiest way to tell, is simply to test it:



As expected, for such small table index was not used.
After some test with various row counts, I found the exact threshold for this particular page to switch to index scan:

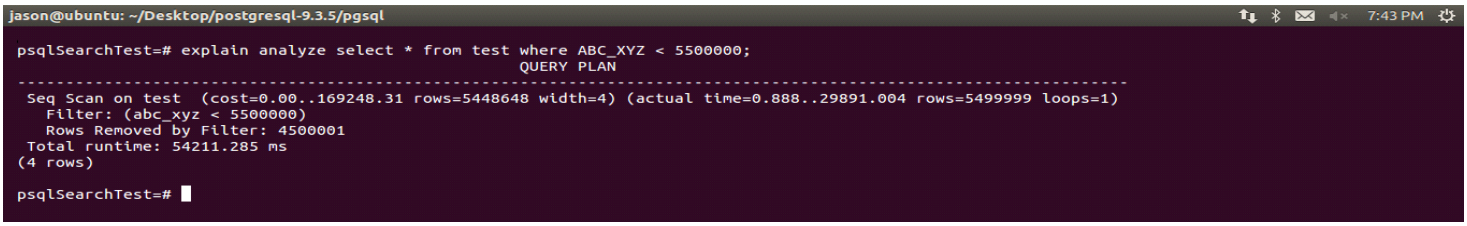


That was the first situation where PostgreSQL chose not to use index.

SITUATION 2

Random reads from disk are much more expensive than sequential ones (due to seeks). This means that if we are fetching more than *some* percent of the page, usage of the index is not sensible. Let's test this threshold. For this I'll need much larger test table:

With this in mind let's force index scan and test 55%:



Time for index scan is actually a bit faster, but not really much. To fix this problem, sequential scan should be chosen, (perhaps at 60%).

So, right now we know that index will not be used if: (1) table is too small or (2) if we get too many records (as percent of the table getting 7million rows from 10 million rows is faster with seq scan, but getting the same 7 million rows from 1 billion row table should use index).

SITUATION 3

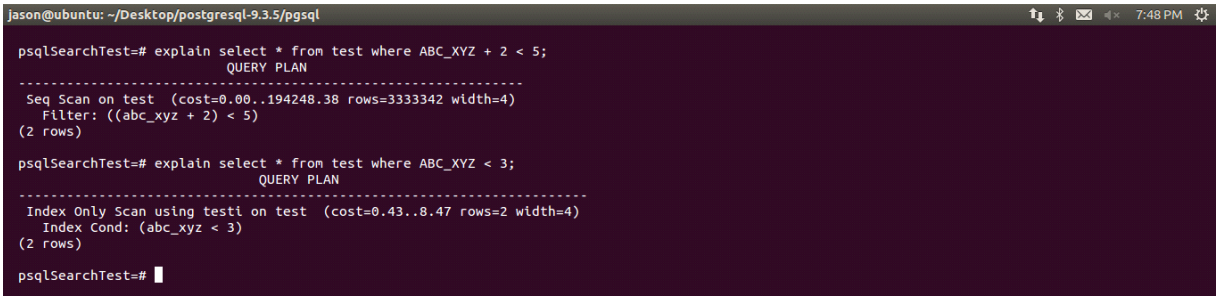
Last situation is simple to explain, but not always simple to understand.
Let's assume that we have:

select * **from** test **where** whatever + 2 < 5;

That's the same as:

select * **from** test **where** whatever < 3;

Right? No its different!



Why is that so?

Operations are done using functions. So basically whatever + 2″ is the same as int4pl(whatever, 2) i.e. call to function int4pl, with 2 arguments.

PostgreSQL doesn't know what which function does. So, knowing that name of the function is int4pl, doesn't mean anything to it, so it can't modify the query to the later form, and it has to assume that int4pl() function can return anything. Which means we can't use index on (column) for search for functionname(column)!

This means that if you want your index to be used you have to keep the indexed operation alone on it's side of comparison operator.

SITUATION 4

Another situation is that not all operations are indexable. For the very simple case like operator, with %' in the beginning of pattern is not indexable (%ABC_XYZ%' you can't use index).
So, when using special data types, or non-trivial operators always check if it's at all possible to index it.

About analyze command

PostgreSQL contains some statistics about values in your table. These statistics are updated with analyze command (either run by hand, or via autovacuum). If these statistics are bad or missing you can get bad results.