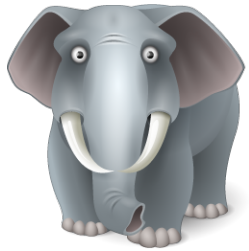
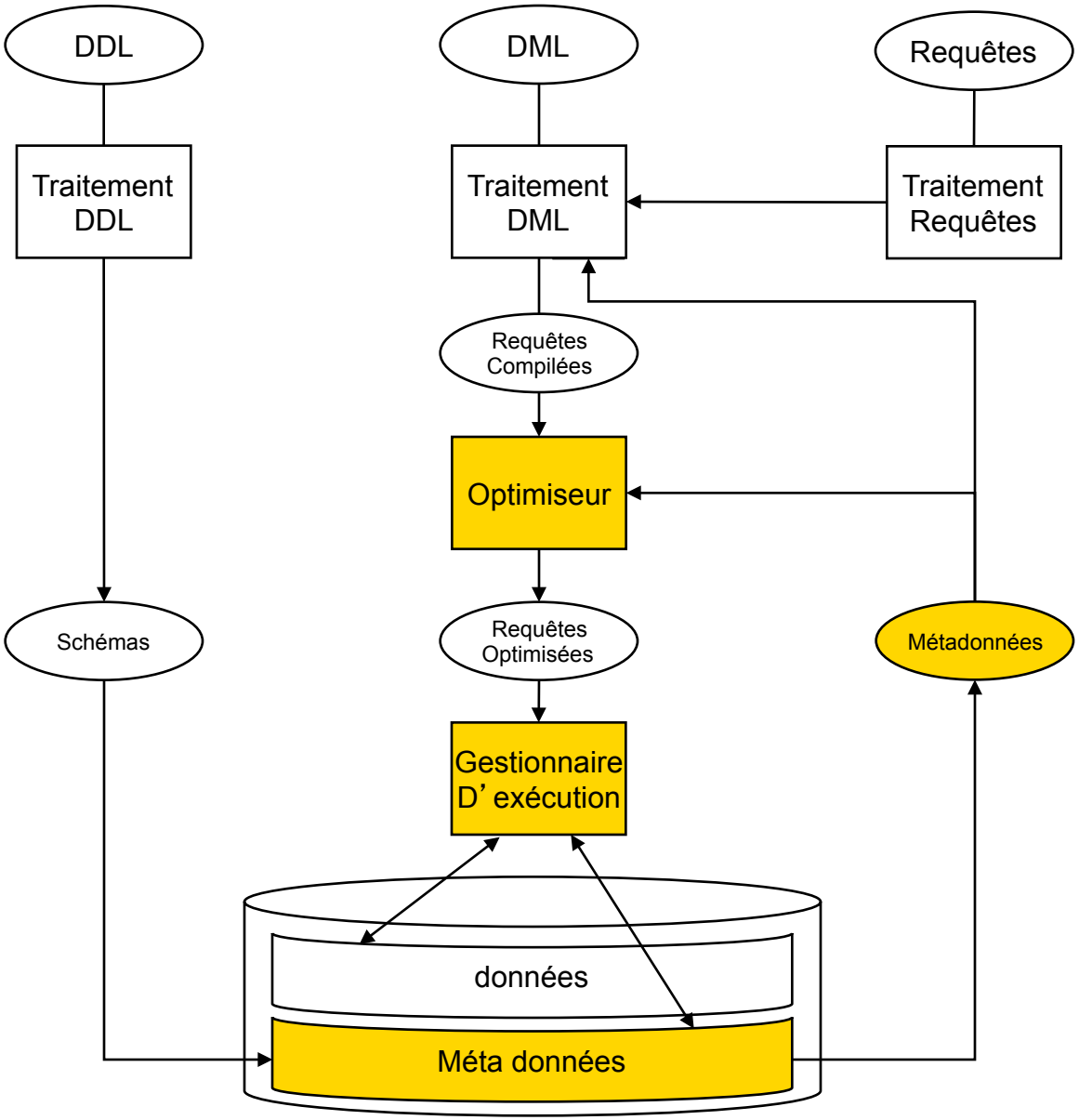
Conception Avancée de Bases de Données

explain Costs computation





D' après C.J DATE

DDL : langage de définition des données; DML : langage de manipulation des données

Postgresql documentation

- http://www.postgresql.org/docs/9.0/static/sql-explain.html

- EXPLAIN displays the execution plan that the PostgreSQL planner generates for the supplied statement.

- The execution plan shows how the table(s) referenced by the statement will be scanned — by plain sequential scan, index scan, etc. — and if multiple tables are referenced, what join algorithms will be used to bring together the required rows from each input table.

FORMAT parameter

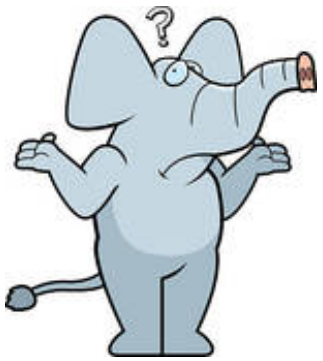
- FORMAT Specify the output format, which can be TEXT, XML, JSON, or YAML.
- Non-text output contains the same information as the text output format, but is easier for programs to parse.
- This parameter defaults to TEXT.

Postgresql documentation : using explain

- http://www.postgresql.org/docs/9.0/static/using-explain.html

- PostgreSQL devises a *query plan* for each query it receive

- PostgreSQL complex *planner* tries to choose good plans.



Jointure entre R et S

```
postgres=# explain select * from r,s;
                        QUERY PLAN
-------------------------------------------------------------------
 Nested Loop  (cost=15.28..4638.08 rows=230400 width=276)
   ->  Seq Scan on r  (cost=0.00..14.80 rows=480 width=138)
   ->  Materialize  (cost=15.28..20.08 rows=480 width=138)
         ->  Seq Scan on s  (cost=0.00..14.80 rows=480 width=138)
(4 lignes)
```

query plan structure

- The structure of a query plan is a tree of *plan nodes*.

- Nodes at the bottom level of the tree are table scan nodes:
  - they return raw rows from a table.

```
postgres=# explain select * from r,s;
                          QUERY PLAN
------------------------------------------------------------------
Nested Loop  (cost=15.28..4638.08 rows=230400 width=276)
  ->  Seq Scan on r  (cost=0.00..14.80 rows=480 width=138)
  ->  Materialize  (cost=15.28..20.08 rows=480 width=138)
        ->  Seq Scan on s  (cost=0.00..14.80 rows=480 width=138)
(4 lignes)
```

query plan structure

- The output of EXPLAIN has one line for each node in the plan tree, showing the basic node type plus the cost estimates that the planner made for the execution of that plan node.

- The first line (topmost node) has the estimated total execution cost for the plan; it is this number that the planner seeks to minimize.

```
postgres=# explain select * from r,s;
                          QUERY PLAN
------------------------------------------------------------------
Nested Loop  (cost=15.28..4638.08 rows=230400 width=276)
  ->  Seq Scan on r  (cost=0.00..14.80 rows=480 width=138)
  ->  Materialize  (cost=15.28..20.08 rows=480 width=138)
        ->  Seq Scan on s  (cost=0.00..14.80 rows=480 width=138)
(4 lignes)
```

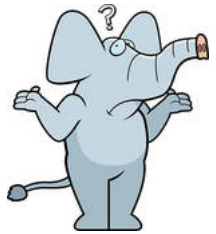
costs

- The costs are measured in arbitrary units determined by the planner's cost parameters

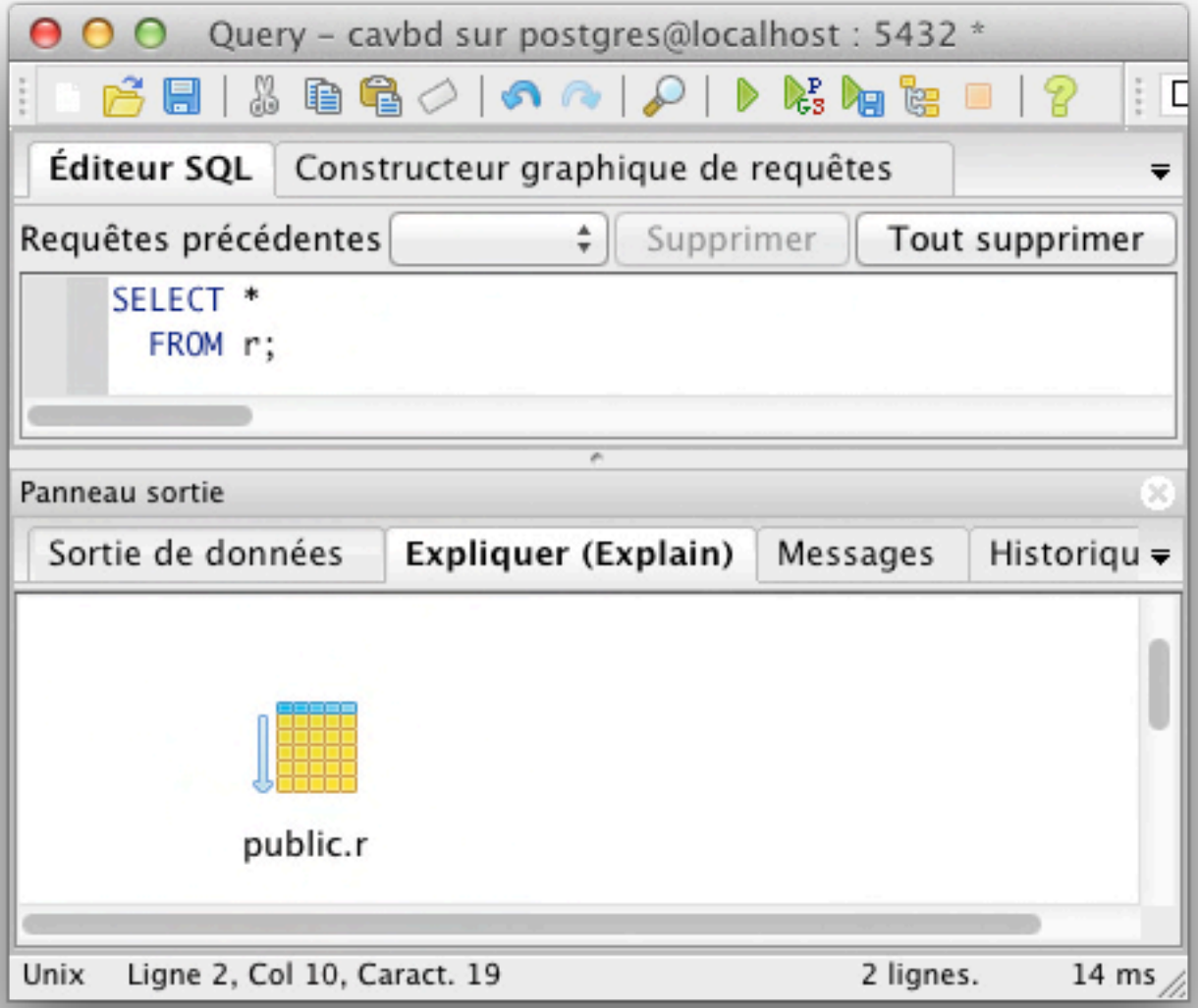
- Traditional practice is to measure the costs in units of disk page fetches; that is, seq_page_cost is conventionally set to 1.0 and the other cost parameters are set relative to that.

One node query plan

```
emmanuelfuchs — psql — 65×7
cavbd=> explain SELECT * FROM r;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on r  (cost=0.00..14425.00 rows=1000000 width=4)
(1 row)

cavbd=>
```



On node PG Admin III graphical query plan



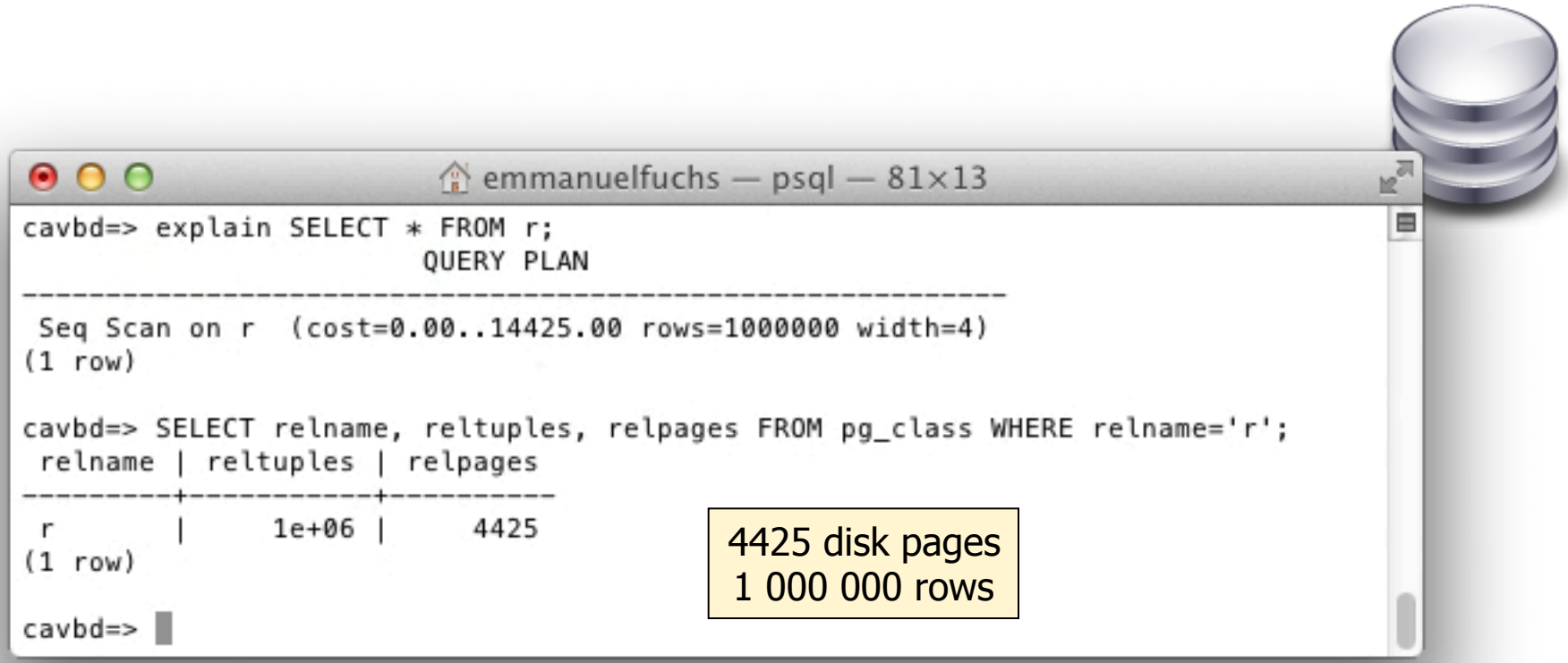
R creation

emmanuelfuchs — psql — 63×5

```
cavbd=> CREATE TABLE r (ri integer);
CREATE TABLE
cavbd=>
```

emmanuelfuchs — psql — 87×5

```
cavbd=> CREATE TABLE r (ri integer);
CREATE TABLE
cavbd=> INSERT INTO r (ri) SELECT round(random()*10) FROM generate_series(1, 1000000);
INSERT 0 1000000
cavbd=>
```



The estimated cost is computed as :

(disk pages read * seq_page_cost) + (rows scanned * cpu_tuple_cost)

postgresql.conf

```
# - Planner Cost Constants -

#seq_page_cost          = 1.0       # measured on an arbitrary scale
#random_page_cost       = 4.0       # same scale as above
#cpu_tuple_cost         = 0.01      # same scale as above
#cpu_index_tuple_cost   = 0.005     # same scale as above
#cpu_operator_cost      = 0.0025    # same scale as above
```

```
seq_page_cost = 1.0

cpu_tuple_cost = 0.01
```

/Library/PostgreSQL/9.3/share/postgresql/postgresql.conf.sample

estimated cost

emmanuelfuchs — psql — 81×13

```
cavbd=> explain SELECT * FROM r;
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on r  (cost=0.00..14425.00 rows=1000000 width=4)
(1 row)

cavbd=> SELECT relname, reltuples, relpages FROM pg_class WHERE relname='r';
 relname | reltuples | relpages
---------+-----------+----------
 r       |     1e+06 |     4425
(1 row)

cavbd=>
```

seq_page_cost = 1.0

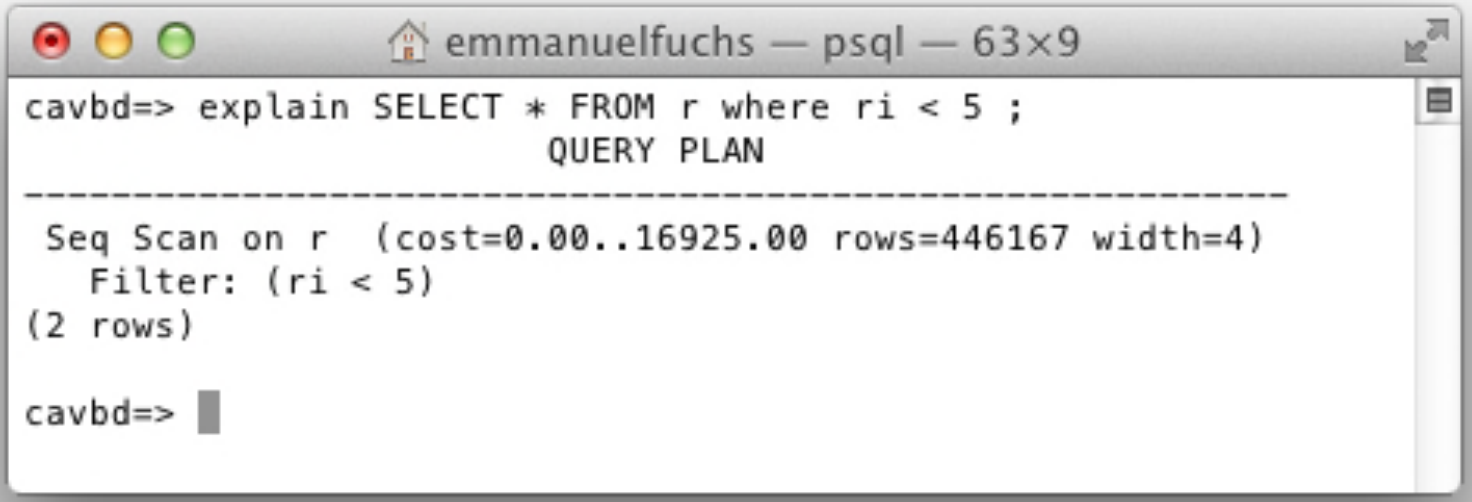
cpu_tuple_cost = 0.01

4425 disk pages
1 000 000 rows

The estimated cost is computed as :

(disk pages read * seq_page_cost) + (rows scanned * cpu_tuple_cost)
4425 * 1 + 1 000 000 * 0.01
4425 + 10 000 = 14 425

Select with where clause



The estimate of output rows has been reduced because of the WHERE clause.

However the cost increses due to the CPU time spent to check the where clause.

1 000 000 * cpu_operator_cost = 1 000 000 * 0, 0025 = 16 925

Node Costs addition

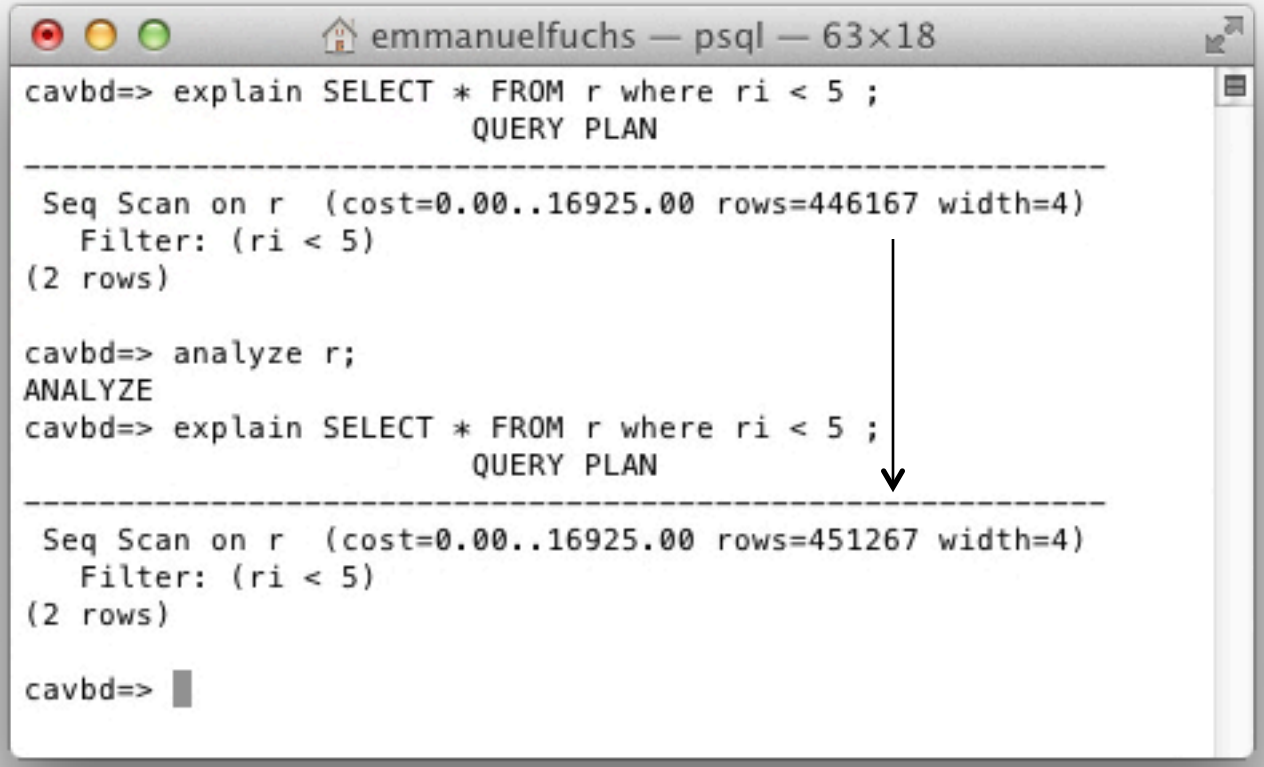
emmanuelfuchs — psql — 61×6

```
cavbd=> EXPLAIN SELECT * FROM country;
 Seq Scan on country  (cost=0.00..7.39 rows=239 width=113)

cavbd=>
```

emmanuelfuchs — psql — 78×6

```
cavbd=> EXPLAIN SELECT * FROM country WHERE population > 100000000;
 Seq Scan on country  (cost=0.00..7.99 rows=10 width=113)
   Filter: (population > 100000000)

cavbd=>
```

Analyze effect



Analyze effect

emmanuelfuchs — psql — 60×27

```
cavbd=> explain SELECT * FROM r where ri < 5 ;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on r  (cost=0.00..16925.00 rows=446167 width=4)
   Filter: (ri < 5)
(2 rows)

cavbd=> analyze r;
ANALYZE
cavbd=> explain SELECT * FROM r where ri < 5 ;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on r  (cost=0.00..16925.00 rows=451267 width=4)
   Filter: (ri < 5)
(2 rows)

cavbd=> analyze r;
ANALYZE
cavbd=> explain SELECT * FROM r where ri < 5 ;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on r  (cost=0.00..16925.00 rows=448500 width=4)
   Filter: (ri < 5)
(2 rows)

cavbd=>
```

Where clause selectivity and index

emmanuelfuchs — psql — 89×11

```
cavbd=> EXPLAIN SELECT * FROM employees WHERE employee_id > 10;
                            QUERY PLAN
------------------------------------------------------------------
 Seq Scan on employees  (cost=0.00..155.50 rows=990 width=1031)
   Filter: (employee_id > 10::numeric)
(2 rows)

cavbd=>
```

emmanuelfuchs — psql — 87×9

```
cavbd=> EXPLAIN SELECT * FROM employees WHERE employee_id > 1000;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Index Scan using employees_pk on employees  (cost=0.28..4.33 rows=1 width=1031)
   Index Cond: (employee_id > 1000::numeric)
(2 rows)

cavbd=>
```

SQL WHERE condition

- If no SQL “WHERE” condition the DBMS Request Planner chooses a seq scan.
- If WHERE condition is not selective the Planner keep using seq scan
- If the WHERE condition is selective enough, the Request Planner may switch completely to an index scan.