



PA036: DB Project

1. PostgreSQL

Vlastislav Dohnal

Relational DBMS

■ PostgreSQL

- open-source relational database management system
- ver. 16 – [docs](#)

■ Accessing DB

- a command-line client: [psql](#)

■ Benchmarking DB

- a command-line tool: [pgbench](#)

Query Plan

■ command EXPLAIN

□ shows how a database will execute a query

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 7000;  
                QUERY PLAN
```

```
-----  
Seq Scan on tenk1 (cost=0.00..483.00 rows=7001 width=244)  
  Filter: (unique1 < 7000)
```

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 100;
```

```
                QUERY PLAN
```

```
-----  
Bitmap Heap Scan on tenk1 (cost=5.07..229.20 rows=101 width=244)  
  Recheck Cond: (unique1 < 100)  
-> Bitmap Index Scan on tenk1_unique1 (cost=0.00..5.04 rows=101 width=0)  
    Index Cond: (unique1 < 100)
```

Query Plan (cont.)

```
EXPLAIN SELECT * FROM tenk1 t1, tenk2 t2
  WHERE t1.unique1 < 100 AND t1.unique2 = t2.unique2;
```

QUERY PLAN

```
Hash Join (cost=230.47..713.98 rows=101 width=488)
  Hash Cond: (t2.unique2 = t1.unique2)
  -> Seq Scan on tenk2 t2 (cost=0.00..445.00 rows=10000 width=244)
  -> Hash (cost=229.20..229.20 rows=101 width=244)
      -> Bitmap Heap Scan on tenk1 t1
          (cost=5.07..229.20 rows=101 width=244)
          Recheck Cond: (unique1 < 100)
          -> Bitmap Index Scan on tenk1_unique1
              (cost=0.00..5.04 rows=101 width=0)
              Index Cond: (unique1 < 100)
```

Query Plan (cont.)

- Add analyze to show real execution times

```
EXPLAIN ANALYZE SELECT * FROM tenk1
    WHERE unique1 < 100 AND unique2 > 9000 LIMIT 2;
```

QUERY PLAN

```
Limit (cost=0.29..14.71 rows=2 width=244)
    (actual time=0.177..0.249 rows=2 loops=1)
-> Index Scan using tenk1_unique2 on tenk1
    (cost=0.29..72.42 rows=10 width=244)
    (actual time=0.174..0.244 rows=2 loops=1)
    Index Cond: (unique2 > 9000)
    Filter: (unique1 < 100)
    Rows Removed by Filter: 287
Planning time: 0.096 ms
Execution time: 0.336 ms
```

It may signal
inefficiency of filter.

Planner may show
discrepancy in the
number of rows

If planner is wrong in estimates, try to update statistics – vacuum command.

Query Plan (cont.)

- Beware of data modifying queries under inspection -> use transactions

```
BEGIN;  
EXPLAIN ANALYZE UPDATE tenk1 SET hundred = hundred + 1 WHERE unique1 < 100;  
QUERY PLAN
```

```
-----  
Update on tenk1 (cost=5.08..230.08 rows=0 width=0)  
                (actual time=3.791..3.792 rows=0 loops=1)  
-> Bitmap Heap Scan on tenk1 (cost=5.08..230.08 rows=102 width=10)  
    (actual time=0.069..0.513 rows=100 loops=1)  
    Recheck Cond: (unique1 < 100)  
    Heap Blocks: exact=90  
-> Bitmap Index Scan on tenk1_unique1  
    (cost=0.00..5.05 rows=102 width=0)  
    (actual time=0.036..0.037 rows=300 loops=1)  
    Index Cond: (unique1 < 100)
```

```
Planning Time: 0.113 ms
```

```
Execution Time: 3.850 ms
```

```
ROLLBACK;
```

Query Plan (cont.)

■ Add buffers – shows how much data was read

```
EXPLAIN (ANALYZE, BUFFERS) SELECT * FROM tenk1
        WHERE unique1 < 100 AND unique2 > 9000;
        QUERY PLAN
```

```
-----
Bitmap Heap Scan on tenk1 (cost=25.08..60.21 rows=10 width=244)
    (actual time=0.323..0.342 rows=10 loops=1)
  Recheck Cond: ((unique1 < 100) AND (unique2 > 9000))
  Buffers: shared hit=15
-> BitmapAnd (cost=25.08..25.08 rows=10 width=0)
    (actual time=0.309..0.309 rows=0 loops=1)
  Buffers: shared hit=7
-> Bitmap Index Scan on tenk1_unique1
    (cost=0.00..5.04 rows=101 width=0)
    (actual time=0.043..0.043 rows=100 loops=1)
  Index Cond: (unique1 < 100)
  Buffers: shared hit=2
-> Bitmap Index Scan on tenk1_unique2
    (cost=0.00..19.78 rows=999 width=0)
    (actual time=0.227..0.227 rows=999 loops=1)
  Index Cond: (unique2 > 9000)
  Buffers: shared hit=5
```

Planning time: 0.088 ms

Execution time: 0.423 ms

Query Rewriting

- Modifying the query text to eliminate unnecessary operations
 - Keeping the result identical!!!
- Techniques
 - Use of indexes
 - attribute selection, inclusion of attributes, clustering the table by an index
 - Elimination unnecessary ops (DISTINCT, GROUP BY, ...)
 - (Correlated) subqueries
 - changing subquery to a join, “with” clause
 - Temporary tables
 - Incorrect use of having
 - Materialized views and its maintenance

Statistics in PostgreSQL

■ Relation hotel

Statistic	Value
Sequential Scans	4
Sequential Tuples Read	500
Index Scans	1
Index Tuples Fetched	500
Tuples Inserted	500
Tuples Updated	0
Tuples Deleted	0
Tuples HOT Updated	0
Live Tuples	500
Dead Tuples	0
Heap Blocks Read	5
Heap Blocks Hit	514
Index Blocks Read	4
Index Blocks Hit	599
Toast Blocks Read	
Toast Blocks Hit	
Toast Index Blocks Read	
Toast Index Blocks Hit	
Last Vacuum	
Last Autovacuum	
Last Analyze	
Last Autoanalyze	2010-04-15 13:52:03.54614+02
Table Size	40 kB
Toast Table Size	none
Indexes Size	32 kB

Statistics in PostgreSQL

■ Attribute hotel.id








Statistic	Value
Null Fraction	0
Average Width	4
Distinct Values	-1
Most Common Values	
Most Common Frequencies	
Histogram Bounds	{1,50,100,150,200,250,300,350,400,450,500}
Correlation	1

■ Attribute hotel.name








Statistic	Value
Null Fraction	0
Average Width	9
Distinct Values	-1
Most Common Values	
Most Common Frequencies	
Histogram Bounds	{street1,street143,street189,street233,street279,street323,street369,street413,street459,street53,street99}
Correlation	-0.117997

Statistics in PostgreSQL

■ Attribute hotel.state

Properties	Statistics	Dependencies	Dependents
Statistic		Value	
 Null Fraction		0	
 Average Width		7	
 Distinct Values		50	
 Most Common Values		{state32,state8,state14,state36,state42,state48,state6,state16,state30,state47}	
 Most Common Frequencies		{0.038,0.03,0.028,0.028,0.028,0.028,0.028,0.026,0.026,0.026}	
 Histogram Bounds		{state1,state12,state18,state21,state25,state29,state34,state4,state44,state5,state9}	
 Correlation		-0.00743129	

■ Attribute hotel.distance_to_center

Properties	Statistics	Dependencies	Dependents
Statistic		Value	
 Null Fraction		0	
 Average Width		4	
 Distinct Values		10	
 Most Common Values		{6,7,10,3,9,8,2,1,4,5}	
 Most Common Frequencies		{0.108,0.108,0.108,0.106,0.102,0.098,0.096,0.094,0.092,0.088}	
 Histogram Bounds			
 Correlation		0.102588	

Indexes in PostgreSQL

- Index types in [docs](#)

- B-tree, Hash

- GiST, SP-GiST

- for several two-dimensional geometric data types,
 - supports “nearest neighbors” queries

- GIN

- inverted file, for indexing arrays

- BRIN (Block Range Index)

- stores summaries about the values stored in consecutive physical block ranges of a table, good for values well-correlated with the physical order of the table rows

Indexes in PostgreSQL (cont.)

- Multicolumn indexes
 - Mind the order of attributes
- Expression based indexes
 - Evaluate a function to be stored in the index
- Partial indexes
 - Index only a subset of rows

- 
- That' all, folks.