universität innsbruck

# Architektur und Implementierung von Datenbanksystemen

Task 8

Team 3 - Gründlinger Diana, Huber Marcel, Klotz Thomas, Targa Aaron, Thalmann Matthias

# PostgreSQL cost estimation

- Each parameter can be customized
- ANALYZE collects statistics about contents of tables → *pg_statistic*

  → statistics used by query planner (as partially seen in lecture 08: Optimization 2)

- ANALYZE is automatically done by the *autovacuum deamon*
- *pg_statistic:*

  - entry for each column

  - fraction of null values, avg. stored bytes, amount of distinct values, …

# Cost estimation parameters (1)

- Arbitrary scale
- Relative values
- Default: relative to *seq_page_cost*

| Parameter | Default value | Description |
|---|---|---|
| seq_page_cost | 1.0 | Cost of a disk page fetch (sequentially) |
| random_page_cost | 4.0 | Cost of a disk page fetch (non-sequentially) |
| cpu_tuple_cost | 0.01 | Cost of processing each row during a query |
| cpu_index_tuple_cost | 0.005 | Cost of processing each index entry during an index scan |
| cpu_operator_cost | 0.0025 | Cost of processing each operator executed during a query |

# Cost estimation parameters (2)

| Parameter | Default value | Description |
|---|---|---|
| parallel_setup_cost | 1000 | Cost of launching parallel worker processes |
| parallel_tuple_cost | 0.1 | Cost of transferring a tuple from one process to another one |
| min_parallel_table_scan_size | 8MB | Min. amount of table data needed to be scanned for a parallel scan to be considered |
| min_parallel_index_scan_size | 512KB | Min. amount of index data needed to be scanned for a parallel scan to be considered |
| effective_cache_size | 4GB | Assumed amount of disk cache avail. to a single query |
| jit_above_cost | 100,000 | Cost, above which JIT compilation is activated |
| jit_inline_above_cost | 500,000 | Cost, above which JIT comp. attempts inlining |
| jit_optimize_above_cost | 500,000 | Cost, above which JIT comp. applies expensive optimization |

# Cost estimation – Sequential scan

- (disk_page_reads ∗ **seq_page_cost**) + (rows_scanned ∗ **cpu_tuple_cost**)
- Filtering (*WHERE*-clause) does only increase the cost, since each row has to be scanned *and* evalutated

  → + (rows_scanned * **cpu_operator_cost**)

**Example 1:**
*tenk1* has 358 disk pages and 10,000 rows

```
EXPLAIN SELECT * FROM tenk1;

                        QUERY PLAN
-----------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..458.00 rows=10000
```

→ $(358 * 1.0) + (10000 * 0.01) = 458.0$

**Example 2:**
*tenk1* has 358 disk pages and 10,000 rows

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

                        QUERY PLAN
-----------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..483.00 rows=7001 width=24
   Filter: (unique1 < 7000)
```

→ $(358 * 1.0) + (10000 * 0.01)$
$+(10000 * 0.0025) = 483.0$

# Cost estimation – Index scan (1)

- Very complex method in *costsize.c*
- Heavily dependent on the implementation of the index (e.g. B-Tree index)
- *amcostestimate* – function-interface; implementation specific for each type of index
- Estimates the total cost of the index (and other costs)

  → depending on type of index and index statistics

  → uses the *genericcostestimate* function

  (takes for example caching into account)

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 = 42;

                                  QUERY PLAN
-------------------------------------------------------------------------
 Index Scan using tenk1_unique1 on tenk1  (cost=0.29..8.30 rows=1 w
   Index Cond: (unique1 = 42)
```

# Cost estimation – Index scan (2) – Example for a B-Tree index

```
WITH costs(idx_cost, tbl_cost) AS (
  SELECT
    ( SELECT round(
        current_setting('random_page_cost')::real * pages +
        current_setting('cpu_index_tuple_cost')::real * tuples +
        current_setting('cpu_operator_cost')::real * tuples
      )
      FROM (
        SELECT relpages * 0.0630 AS pages, reltuples * 0.0630 AS tuples
        FROM pg_class WHERE relname = 'bookings_pkey'
      ) c
    ),
    ( SELECT round(
        current_setting('seq_page_cost')::real * pages +
        current_setting('cpu_tuple_cost')::real * tuples
      )
      FROM (
        SELECT relpages * 0.0630 AS pages, reltuples * 0.0630 AS tuples
        FROM pg_class WHERE relname = 'bookings'
      ) c
    )
)
SELECT idx_cost, tbl_cost, idx_cost + tbl_cost AS total FROM costs;
```

Amount of rows

```
SELECT round(132999::numeric/reltuples::numeric, 4)
FROM pg_class WHERE relname = 'bookings';

 round
--------
 0.0630
(1 row)
```

Source:
https://postgrespro.com/blog/pgsql/5969493

# References


- https://github.com/postgres/postgres/blob/master/src/backend/optimizer/path/costsize.c
- http://morningcoffee.io/the-postgresql-query-cost-model.html
- https://www.postgresql.org/docs/current/runtime-config-query.html
- https://www.postgresql.org/docs/current/using-explain.html
- https://www.postgresql.org/docs/current/index-cost-estimation.html
- https://www.postgresql.org/docs/current/sql-analyze.html
- https://www.postgresql.org/docs/current/routine-vacuuming.html
- https://git.uibk.ac.at/informatik/dbis/dbis-teaching/archimpl-course-material-2022/-/blob/main/slides/08_optimization2.pdf
- https://postgrespro.com/blog/pgsql/5969493
- https://stackoverflow.com/a/57279653/11028838


www.uibk.ac.at