University of Edinburgh
School of Informatics
INFR11199 - Advanced Database Systems (Spring 2024)

Tutorial Sheet 4 - Playing with PostgreSQL

The purpose of this practical sheet is to familiarise you with the query execution engine of PostgreSQL. In particular, you will analyse a few queries and answer questions regarding their performance when turning different knobs of the execution engine. To answer the questions, you might find the following documentation links useful:

- Documentation of `EXPLAIN ANALYZE`:
  `https://www.postgresql.org/docs/14/sql-explain.html`.
- Making sense of the `EXPLAIN ANALYZE` output:
  `https://www.postgresql.org/docs/14/performance-tips.html`.
- PostgreSQL query planner documentation:
  `https://www.postgresql.org/docs/14/runtime-config-query.html`.
- How to create an index:
  `https://www.postgresql.org/docs/14/sql-createindex.html`.
- The system table `pg_class`:
  `https://www.postgresql.org/docs/current/catalog-pg-class.html`.

Prerequisites:

- Install PostgreSQL on your machine and start a PostgreSQL server (plenty of instructions online on how to do this, e.g., `http://postgresguide.com/setup/install.html`; any version will work). Make sure the command-line tool `psql` is working and you can use it to create tables and run queries.
- Download the bay-area-bike-sharing dataset from the course webpage. Unzip the archive and import the data into PostgreSQL using the provided scripts (e.g., by typing the command `psql < import.sql`).

1. **EXPLAIN and ANALYZE**

For the following questions consider the query below:

```
SELECT * FROM trip WHERE bike_id = 10;
```

(a) Provide the PostgreSQL execution plan of the query and the SQL statement you use to generate the result.

Solution:

```
EXPLAIN ANALYZE SELECT * FROM trip WHERE bike_id = 10;

QUERY PLAN
-------------------------------------------------------------------------
Gather  (cost=1000.00..14191.37 rows=790 width=80) (actual time
    =2.556..143.164 rows=248 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on trip  (cost=0.00..13112.37 rows=329 width=80)
       (actual time=4.328..127.060 rows=83 loops=3)
         Filter: (bike_id = 10)
         Rows Removed by Filter: 223237
 Planning Time: 0.824 ms
 Execution Time: 143.286 ms
```

Based on the execution plan:

i. What was the estimated cost (in arbitrary units)? **14191.87**

ii. What was the total runtime (in ms)? **143.286**

iii. What was the estimated number of tuples to be output? **790**

iv. What was the actual number of output tuples? **248**

(b) Create an index on the attribute `bike_id` on the table `trip`. Provide the SQL statement for that and the new execution plan of the query.

Solution:

```
CREATE INDEX idx_bike_id ON trip(bike_id);

QUERY PLAN
-------------------------------------------------------------------------
 Bitmap Heap Scan on trip  (cost=18.55..2424.94 rows=790 width=80) (actual
      time=0.432..1.039 rows=248 loops=1)
   Recheck Cond: (bike_id = 10)
   Heap Blocks: exact=234
   ->  Bitmap Index Scan on idx_bike_id  (cost=0.00..18.35 rows=790 width
       =0) (actual time=0.335..0.335 rows=248 loops=1)
         Index Cond: (bike_id = 10)
 Planning Time: 6.332 ms
 Execution Time: 1.154 ms
```

Based on the execution plan:

i. What was the estimated cost (in arbitrary units)? **2424.94**

ii. What was the total runtime (in ms)? **1.154**

(c) Use the table `pg_class` to answer the following questions.

i. How many pages are used to store the index you created on table `trip`? Provide the answer and the query you use to generate the answer.

Solution:

```
SELECT relpages FROM pg_class
WHERE relname = 'idx_bike_id';

relpages
----------
     1840
```

ii. How many tuples are in the index you created on column `bike_id`? Provide the answer and the query you use to generate the answer.

Solution:

```
SELECT reltuples FROM pg_class
WHERE relname = 'idx_bike_id';

 reltuples
-----------
    669959
```

iii. How many tuples are in the table weather, according to pg_class?

Solution:

```
SELECT reltuples FROM pg_class
WHERE relname = 'weather';

 reltuples
-----------
      3665
```

iv. In the table weather, delete all records of which date is earlier than '2013-10-01'. Provide the SQL statement you use.

Solution:

```
DELETE FROM weather WHERE date < '2013-10-01';
```

v. After deletion, rerun your query from step 3. Is the new result equal to the result of running SELECT COUNT(*) FROM weather?

○ Yes ✓ **No**

vi. ANALYZE is a Postgres function used to collect statistics about a database. You want to use it especially after considerable number of modifications happen to that database. Run ANALYZE, and then rerun your query from step 3 again. Is the new result equal to the result of running SELECT COUNT(*) FROM weather?

✓ **Yes** ○ No

2. **Using Indexes**

In this question, we will learn the conditions under which indexes may or may not be used by the query optimizer.

(a) Create an index on the column start_station_name on the table trip. Provide the SQL command you use.

Solution:

```
CREATE INDEX idx_start_sta_name ON trip ( start_station_name );
```

(b) For each of those queries, answer Yes if the index you created on trip.start_station_name was used in the execution plan, or No otherwise:

i.
```
SELECT * FROM trip
 WHERE start_station_name like 'San';
```

√ **Yes** ○ No

ii.
```
SELECT * FROM trip
  WHERE start_station_name like '%San';
```

○ Yes √ **No**

iii.
```
SELECT * FROM trip
  WHERE start_station_name BETWEEN 'San␣Francisco' AND 'San␣Jose'
    AND end_station_name > 'San';
```

√ **Yes** ○ No

iv.
```
SELECT * FROM trip
  WHERE start_station_name BETWEEN 'San␣Francisco' AND 'San␣Jose'
     OR end_station_name > 'San';
```

○ Yes √ **No**

(c) Make sure you still have an index on the column `trip.bike_id` (you can verify this using `\di` in `psql`). For each of those queries, answer which indexes are used in their execution plans.

i.
```
SELECT * FROM trip
  WHERE start_station_name BETWEEN 'San␣Francisco' AND 'San␣Jose'
    AND bike_id < 10;
```

○ Only the index on `start_station_name` was used.

√ **Only the index on `bike_id` was used.**

○ Both indexes were used.

○ None of the indexes were used.

ii.
```
SELECT * FROM trip
  WHERE start_station_name BETWEEN 'San␣Francisco' AND 'San␣Jose'
    AND bike_id < 500;
```

√ **Only the index on `start_station_name` was used.**

○ Only the index on `bike_id` was used.

○ Both indexes were used.

○ None of the indexes were used.

iii.
```
SELECT * FROM trip
  WHERE start_station_name BETWEEN 'San␣Francisco' AND 'San␣Jose'
    AND bike_id BETWEEN 500 AND 510;
```

○ Only the index on `start_station_name` was used.

○ Only the index on `bike_id` was used.

√ **Both indexes were used.**

○ None of the indexes were used.

iv.
```
SELECT * FROM trip
  WHERE start_station_name > 'San␣Francisco'
    AND bike_id < 500;
```

○ Only the index on start_station_name was used.
○ Only the index on bike_id was used.
○ Both indexes were used.
√ **None of the indexes were used.**

(d) Answer the questions below for the query:

```
SELECT * FROM trip
 WHERE bike_id BETWEEN 10 AND 20;
```

i. Was the index on bike_id used? √ **Yes** ○ No

ii. What percentage of the total records in the table trip was returned? Provide a percent and retain two significant figures. 0.54% (3594 **out of** 669959)

(e) Answer the questions below for the query:

```
SELECT * FROM trip
 WHERE bike_id > 10;
```

i. Was the index on bike_id used? ○ Yes √ **No**

ii. What percentage of the total records in the table trip was returned? Provide a percent and retain two significant figures. 99.92% (669460 **out of** 669959)

(f) Answer the questions below for the query:

```
SELECT * FROM trip
 WHERE bike_id > 10 ORDER BY start_time;
```

i. Which method was used for sorting? **external merge sort**
ii. Where did the sorting happen? ○ Memory √ **Disk**
iii. How much space was used for sorting? 22080**kB per worker**
iv. What was the total runtime (in ms)? 891.222 **ms (any number)**

(g) Display PostgreSQL working memory with SHOW work_mem;. Increase PostgreSQL working memory with the command SET work_mem = '128MB';. For the same query from part vi., answer the following questions:

i. Which method was used for sorting? **quick sort**
ii. Where did the sorting happen? √ **Memory** ○ Disk
iii. How much space was used for sorting? 115, 909**kB**
iv. What was the total runtime (in ms)? 454.680 **ms (any number)**

(h) Execute the command RESET work_mem; to get PostgreSQL working memory back to the default value (or your answers for the next questions will turn out wrong).

3. **Joins**

In this question, we will learn about different methods used by PostgreSQL for executing joins. Make sure you reset `work_mem` to its default value (i.e., `RESET work_mem;`).

Answer the following questions based on the query below:

```sql
SELECT trip.*, station.city
  FROM trip, station
 WHERE trip.start_station_id = station.station_id
   AND bike_id < 200;
```

(a) Provide the query plan for the above query.

Solution:

```
QUERY PLAN
---------------------------------------------------------------------------
 Hash Join  (cost=1093.33..11604.43 rows=58107 width=92) (actual time
     =29.274..85.848 rows=58161 loops=1)
   Hash Cond: (trip.start_station_id = station.station_id)
   ->  Bitmap Heap Scan on trip  (cost=1090.75..11440.09 rows=58107 width
       =80) (actual time=29.108..52.497 rows=58161 loops=1)
         Recheck Cond: (bike_id < 200)
         Heap Blocks: exact=9541
         ->  Bitmap Index Scan on idx_bike_id  (cost=0.00..1076.23 rows
             =58107 width=0) (actual time=25.510..25.510 rows=58161 loops
             =1)
               Index Cond: (bike_id < 200)
   ->  Hash  (cost=1.70..1.70 rows=70 width=14) (actual time=0.126..0.126
       rows=70 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 12kB
         ->  Seq Scan on station  (cost=0.00..1.70 rows=70 width=14) (
             actual time=0.023..0.068 rows=70 loops=1)
 Planning Time: 0.900 ms
 Execution Time: 89.936 ms
```

Based on the execution plan:

i. Which join method was used? **Hash join**

ii. What was the estimated cost (in arbitrary units)? **11604.43**

iii. What was the total runtime (in ms)? **89.936**

(b) Execute the command `SET enable_hashjoin = false;` to disable hash joins. Provide the new query plan.

Solution:

```
QUERY PLAN
---------------------------------------------------------------------------
 Merge Join  (cost=18625.07..19497.02 rows=58107 width=92) (actual time
     =83.404..124.602 rows=58161 loops=1)
   Merge Cond: (trip.start_station_id = station.station_id)
   ->  Sort  (cost=18621.22..18766.49 rows=58107 width=80) (actual time
       =83.322..99.226 rows=58161 loops=1)
```

```
        Sort Key: trip.start_station_id
        Sort Method: external merge  Disk: 5400kB
        ->  Bitmap Heap Scan on trip  (cost=1090.75..11440.09 rows=58107
            width=80) (actual time=15.734..39.091 rows=58161 loops=1)
              Recheck Cond: (bike_id < 200)
              Heap Blocks: exact=9541
              ->  Bitmap Index Scan on idx_bike_id  (cost=0.00..1076.23
                  rows=58107 width=0) (actual time=13.722..13.722 rows
                  =58161 loops=1)
                    Index Cond: (bike_id < 200)
  ->  Sort  (cost=3.85..4.02 rows=70 width=14) (actual time=0.075..0.084
      rows=70 loops=1)
        Sort Key: station.station_id
        Sort Method: quicksort  Memory: 28kB
        ->  Seq Scan on station  (cost=0.00..1.70 rows=70 width=14) (
            actual time=0.016..0.033 rows=70 loops=1)
Planning Time: 0.642 ms
Execution Time: 134.899 ms
```

Based on the execution plan:

i. Which join method was used? **Sort-merge join**

ii. What was the estimated cost (in arbitrary units)? **19497.02**

iii. What was the total runtime (in ms)? **134.899**

(c) Execute the command `SET enable_mergejoin = false;` to disable merge joins. Provide the new query plan.

Solution:

```
QUERY PLAN
------------------------------------------------------------------------
 Nested Loop  (cost=1090.90..20735.16 rows=58107 width=92) (actual time
     =19.123..179.892 rows=58161 loops=1)
   ->  Bitmap Heap Scan on trip  (cost=1090.75..11440.09 rows=58107 width
       =80) (actual time=19.095..43.152 rows=58161 loops=1)
         Recheck Cond: (bike_id < 200)
         Heap Blocks: exact=9541
         ->  Bitmap Index Scan on idx_bike_id  (cost=0.00..1076.23 rows
             =58107 width=0) (actual time=16.637..16.638 rows=58161 loops
             =1)
               Index Cond: (bike_id < 200)
   ->  Index Scan using station_pkey on station  (cost=0.14..0.16 rows=1
       width=14) (actual time=0.002..0.002 rows=1 loops=58161)
         Index Cond: (station_id = trip.start_station_id)
 Planning Time: 0.384 ms
 Execution Time: 185.202 ms
```

Based on the execution plan:

i. Which join method was used? **Nested loops join w/ index scans**

ii. What was the estimated cost (in arbitrary units)? **20735.16**

iii. What was the total runtime (in ms)? **185.202**

(d) Execute the command `SET enable_indexscan = false; SET enable_bitmapscan = false;` to disable index scans. Give the new plan.

Solution:

```
QUERY PLAN
------------------------------------------------------------------------
 Nested Loop  (cost=0.00..78164.76 rows=58107 width=92) (actual time
     =0.101..595.171 rows=58161 loops=1)
   Join Filter: (trip.start_station_id = station.station_id)
   Rows Removed by Join Filter: 1637515
   ->  Seq Scan on trip  (cost=0.00..17997.49 rows=58107 width=80) (actual
        time=0.039..179.165 rows=58161 loops=1)
         Filter: (bike_id < 200)
         Rows Removed by Filter: 611798
   ->  Materialize  (cost=0.00..2.05 rows=70 width=14) (actual time
       =0.000..0.002 rows=29 loops=58161)
         ->  Seq Scan on station  (cost=0.00..1.70 rows=70 width=14) (
             actual time=0.016..0.048 rows=70 loops=1)
 Planning Time: 0.460 ms
 Execution Time: 599.250 ms
```

Based on the execution plan:

i. Which join method was used? **Nested loops join**

ii. What was the estimated cost (in arbitrary units)? **78164.76**

iii. What was the total runtime (in ms)? **599.250**

(e) Execute these commands to re-enable the different joins.

```
RESET enable_mergejoin;
RESET enable_hashjoin;
RESET enable_indexscan;
RESET enable_bitmapscan;
```