University of Trieste

Data Management for Big Data Course

Academic Year 2022–2023

Data Warehouse case study

Davide Capone* Sandro Junior Della Rovere† Enrico Stefanel‡

Last update on 2023–06–23T15:31:15Z.

Contents

*davide.capone@studenti.units.it
†sandrojunior.dellarovere@studenti.units.it
‡enrico.stefanel@studenti.units.it

1 Introduction

The aim of this project is to study an efficient implementation of a suite of business oriented ad-hoc queries over the public TPC-H benchmark, which can be considered as a Big Data database, that has been implemented in Postgres.

1.1 TPC-H benchmark database

The TPC-H benchmark is a decision support benchmark that can be downloaded from the TPC official website. The data generator lets the user specify a *scale factor* in order to control the size of the resulted database. Our choices was to use a *scale factor* of 10, meaning that the overall database size is approximately 13 GB.

1.1.1 Database statistics

The benchmark is composed by eight tables:

- `CUSTOMER`, with 16 columns and 1 500 000 tuples (312 MB);
- `LINEITEM`, with 32 columns and 59 986 052 tuples (11 GB); the main attributes that are going to be used are:
  - `l_extendedprice` (1 351 462 distinct values, i.e. there is an average of 44 tuples with the same value, that range from 900.91 to 104 949.50),
  - `l_discount` (11 distinct values, i.e. there is an average of 5 453 277 tuples with the same value, that range from 0.00 to 0.10),
  - `l_returnflag` (which can assume values `A`→accepted, `R`→returned, `N`→not yet delivered; the percentage of tuples for `A` and `R` are almost 25 %, while the percentage of tuples where `l_returnflag` is `N` about 50 %),
  - `l_commitdate` (2466 distinct values, i.e. there is an average of 24 325 tuples with the same value, that range from 1992-01-31 to 1998-10-31),
  - `l_receiptdate` (2555 distinct values, i.e. there is an average of 23 478 tuples with the same value, that range from 1992-01-03 to 1998-12-31);
- `NATION`, with 8 columns and 25 tuples (24 kB);
- `ORDERS`, with 18 columns and 1 500 000 tuples (2481 kB); the main attributes that are going to be used are:
  - `o_orderdate` (2406 distinct values, i.e. there is an average of 6234 tuples with the same value, that range from 1992-01-01 to 1998-08-02);
- `PART`, with 18 columns and 2 000 000 tuples (363 MB); the main attributes that are going to be used are:
  - `p_type` (150 distinct values, i.e. there is an average of 13 333 tuples with the same value);

- `PARTSUPP`, 10 columns and with 8 000 000 tuples (1535 MB);
- `REGION`, 6 columns and with 5 tuples (24 kB);
- `SUPPLIER`, 14 columns and with 100 000 tuples (20 MB).

Other attributes have been used, but statistics about them have been omitted for lack of usefulness (e.g., keys of the tables, for which the cardinality is exactly the cardinality of the corresponding table).

1.1.2 Database SQL definition

The SQL definition of the tables can be found on the official benchmark download.

1.2 Organization of work

Each of the group component implemented a query (specifically, Della Rovere designed the first query, Capone on the second one, and Stefanel the last one). After regular meetings between the group members comparing the solo works, common substructures between queries have been detected, and the following work proceed in a coral way.

All the execution times reported in the present report are the results of runs on the same machine, with roughly the same external factors. In particular, a computer with an Apple M1 processor, 8 GB of RAM, and macOS operating system has been used.

2 Set of queries

2.1 Export/import revenue value

It is asked to return the *export/import revenue* between two different nations (E, I) where E is the nation of the lineitem supplier and I the nation of the lineitem customer, and where the *revenue* is defined as

```
SUM(l_extendedprice*(1-l_discount))
```

. The aggregation should be performed with the Month → Quarter → Year, (Part) Type and Nation → Region roll-ups.

```
1  WITH lineitem_orders AS (
2      SELECT
3          l_partkey,
4          l_suppkey,
5          o_orderdate,
6          o_custkey,
7          l_extendedprice,
8          l_discount
```

```
    FROM lineitem JOIN orders ON (l_orderkey = o_orderkey)
), customer_location AS (
    SELECT
        c_custkey,
        c_name,
        n_nationkey AS c_nationkey,
        n_name AS c_nationname,
        r_regionkey AS c_regionkey,
        r_name AS c_regionname
    FROM customer
        JOIN nation ON (c_nationkey = n_nationkey)
        JOIN region ON (n_regionkey = r_regionkey)
), supplier_location AS (
    SELECT
        s_suppkey,
        s_name,
        n_nationkey AS s_nationkey,
        n_name AS s_nationname,
        r_regionkey AS s_regionkey,
        r_name AS s_regionname
    FROM supplier
        JOIN nation ON (s_nationkey = n_nationkey)
        JOIN region ON (n_regionkey = r_regionkey)
), query1 AS (
    SELECT
        EXTRACT (YEAR FROM o_orderdate) AS _year,
        EXTRACT (QUARTER FROM o_orderdate) AS _quarter,
        EXTRACT (MONTH FROM o_orderdate) AS _month,
        c_regionname,
        c_nationname,
        c_name,
        s_regionname,
        s_nationname,
        s_name,
        p_type,
        SUM(l_extendedprice * (1 - l_discount)) AS revenue
    FROM lineitem_orders
        JOIN part ON l_partkey = p_partkey
        JOIN supplier_location ON (s_suppkey = l_suppkey)
        JOIN customer_location ON (c_custkey = o_custkey)
    WHERE s_nationkey <> c_nationkey
    GROUP BY
        _year,
```

```
52          _quarter,
53          _month,
54          c_regionkey,
55          c_regionname,
56          c_nationkey,
57          c_nationname,
58          c_custkey,
59          c_name,
60          s_regionkey,
61          s_regionname,
62          s_nationkey,
63          s_nationname,
64          s_suppkey,
65          s_name,
66          p_type
67  )
68  SELECT * FROM query1;
```

2.2 Late delivery

It is asked to retrieve the number of orders where at least one "lineitem" has been received later than the committed date. The aggregation should be performed with the Month → Year roll-up, and the (Customer's) Nation → Region roll-up.

```
1   WITH lineitem_orders AS (
2       SELECT
3           o_orderkey,
4           l_partkey,
5           l_suppkey,
6           o_orderdate,
7           o_custkey,
8           l_commitdate,
9           l_receiptdate
10      FROM lineitem JOIN orders ON (l_orderkey = o_orderkey)
11  ), customer_location AS (
12      SELECT
13          c_custkey,
14          n_nationkey AS c_nationkey,
15          n_name AS c_nationname,
16          r_regionkey AS c_regionkey,
17          r_name AS c_regionname
18      FROM customer
19          JOIN nation ON (c_nationkey = n_nationkey)
```

```
        JOIN region ON (n_regionkey = r_regionkey)
), query2 AS (
SELECT
    EXTRACT(YEAR FROM o_orderdate) AS _year,
    EXTRACT(MONTH FROM o_orderdate) AS _month,
    c_regionname,
    c_nationname,
    COUNT(DISTINCT(o_orderkey)) AS orders_no
FROM lineitem_orders
    JOIN part ON l_partkey = p_partkey
    JOIN customer_location ON (c_custkey = o_custkey)
WHERE
    l_receiptdate > l_commitdate
    -- AND _month = 1
    -- AND p_type = 'PROMO BURNISHED COPPER'
GROUP BY
    _year,
    _month,
    c_regionkey,
    c_regionname,
    c_nationkey,
    c_nationname
)
SELECT * FROM query2;
```

2.3 Returned item loss

It is asked to retrieve the *revenue loss* for customers who might be having problems with the parts that are shipped to them, where a *revenue loss* is defined as

```
SUM(l_extendedprice*(1-l_discount))
```

for all qualifying *lineitems*.

```
WITH lineitem_orders AS (
    SELECT
        o_orderkey,
        o_orderdate,
        o_custkey,
        l_extendedprice,
        l_discount,
        l_returnflag
    FROM lineitem JOIN orders ON (l_orderkey=o_orderkey)
),
```

```
11 query3 AS (
12 SELECT
13     EXTRACT(YEAR FROM o_orderdate) AS _year,
14     EXTRACT(QUARTER FROM o_orderdate) AS _quarter,
15     EXTRACT(MONTH FROM o_orderdate) AS _month,
16     c_name,
17     SUM(l_extendedprice*(1-l_discount)) AS returnloss
18 FROM
19     lineitem_orders
20     JOIN customer ON (o_custkey=c_custkey)
21 WHERE
22     l_returnflag='R'
23     -- AND c_name='Customer#000129976'
24     -- AND EXTRACT(QUARTER FROM o_orderdate) = 1
25 GROUP BY
26     _year,
27     _quarter,
28     _month,
29     c_custkey,
30     c_name
31 )
32 SELECT * FROM query3;
```

2.4 Execution times

The query timings have been measured as previously discussed in subsection 1.2. Furthermore, no consecutive runs have been performed on the same query, in order to reduce external biases (following a Round-Robin schema).

Query	Run 1	Run 2	Run 3	Run 4	Run 5	μ	σ
1	643	685	724	698	813	713	63
2	201	203	205	203	203	203	2
3	98	102	101	102	101	101	2

Table 1: Naïve query timings, in seconds.

3 Materialization

After a study on the given set of queries, some common intermediate results have been detected between the three. In order to try lowering the average execution cost, materialized views have been defined starting from the said results.

```sql
-- used by Q1, Q2, Q3
CREATE MATERIALIZED VIEW lineitem_orders_mv AS
    SELECT
        o_orderkey,
        l_partkey,
        l_suppkey,
        o_orderdate,
        o_custkey,
        l_extendedprice,
        l_discount,
        l_returnflag,
        l_commitdate,
        l_receiptdate
    FROM lineitem JOIN orders ON (l_orderkey = o_orderkey);

-- used by Q1 and Q2
CREATE MATERIALIZED VIEW customer_location_mv AS
    SELECT
        c_custkey,
        c_name,
        n_nationkey AS c_nationkey,
        n_name AS c_nationname,
        r_regionkey AS c_regionkey,
        r_name AS c_regionname
    FROM customer
        JOIN nation ON (c_nationkey = n_nationkey)
        JOIN region ON (n_regionkey = r_regionkey);

-- used by Q1 and Q2
CREATE MATERIALIZED VIEW supplier_location_mv AS
    SELECT
        s_suppkey,
        s_name,
        n_nationkey AS s_nationkey,
        n_name AS s_nationname,
        r_regionkey AS s_regionkey,
        r_name AS s_regionname
    FROM supplier
        JOIN nation ON (s_nationkey = n_nationkey)
        JOIN region ON (n_regionkey = r_regionkey);
```

3.1 Execution times

As for the base versions of queries, we collected statistics on execution timings and results can be observed on Table 2. Further considerations are reported in the section 5.

Query	Run 1	Run 2	Run 3	Run 4	Run 5	μ	σ
1	643	685	724	698	813	713	63
2	201	203	205	203	203	203	2
3	98	102	101	102	101	101	2

Table 2: Query timings with materialized views, in seconds.

4 Indexes design

...

5 Conclusions

Siccome nelle viste materializzate ci sono più attributi rispetto alle tableele intermedie con WITH, nella prima query, allora quella query è più pesante con le viste materializzate.

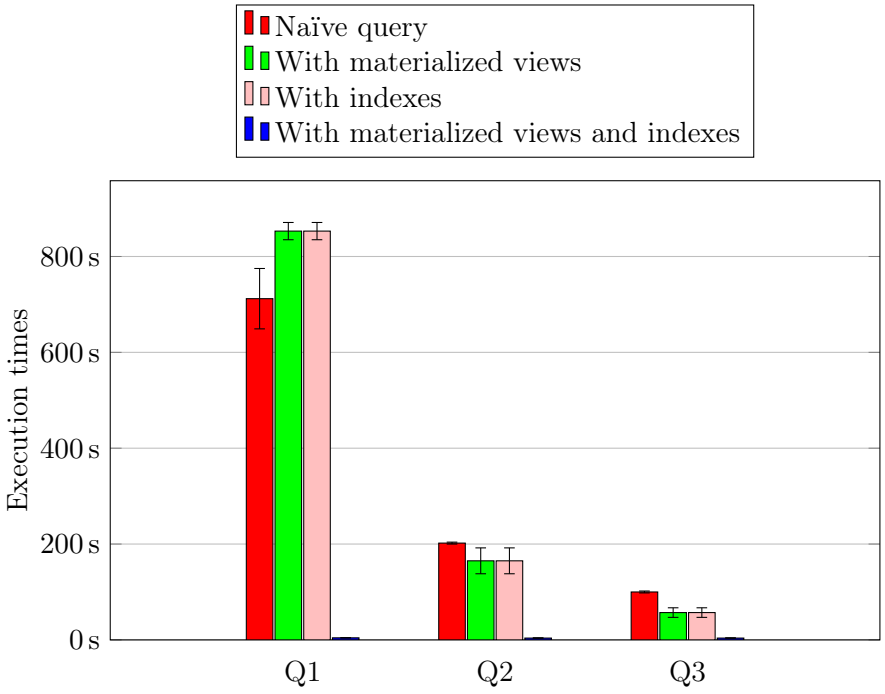

Figure 1: Query timings