Data Management - Final Project

Sara Carpenè, Alessio Valentinis, Marco Zampar

July 18, 2024

1 Introduction

This project has the aim of optimize a set of four queries from the TPC benchmark H The database consists of eight tables: customer, lineitem, nation, orders, part, partsupp, region, supplier. The relations between tables can be seen in the schema in figure 1.

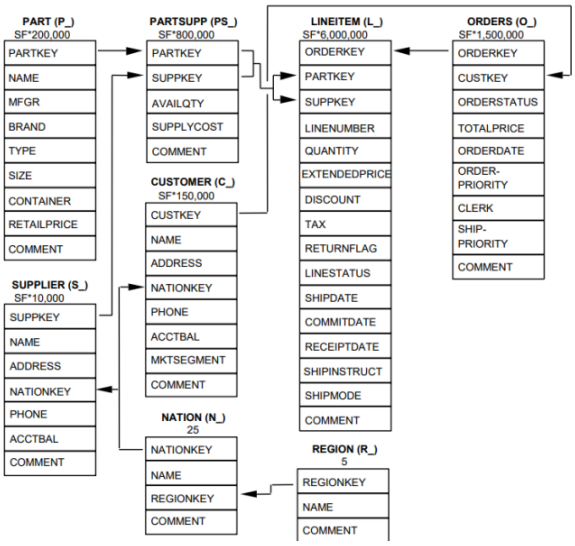

Figure 1: Schema of the relaction between tables in the TPC-H benchmark

1.1 Creation and population of the database

The size of the database is scalable, and depends on a scale factor (SF), and we were given data generated from $SF = 10$. Each table, from the documentation, has its own primary key and one or more foreign keys. Complete description

of the table can be found here, while the complete SQL tables implementation and creation can be found here.

For the purpose of space economy, we opted for an initial "vanilla" creation of the tables, without any kind of keys: this choice is furthermore supported by the fact that we are dealing with a Data Warehousing context, so we expect the data to be already prepared and cleaned from duplicates.

After having populated the tables, we inserted all the keys reported in the documentation in order to work properly with the relations. However, always for the purpose of gaining some space, we decided not to implement the *Primary key* to the Main table `lineitem`, as (for the same purposes described above,) we don't need to check for uniqueness constraints during the population of the Database, and for the sake of optimization, a simple index should help us to spare some pretty useful space.

In the table 1, we provide some information about the dimension of the various tables, in increasing order.

Table	Number of rows	Dim without keys (MB)	Dim with keys (MB)
region	5	0.01	0.02
nation	25	0.01	0.02
supplier	100000	17.35	19.51
customer	1500000	290.17	322.32
part	2000000	320.14	363.00
partsupp	8000000	1362.80	1535.12
orders	15000000	2038.97	2360.30
lineitem	59986052	8787.95	10073.67

Table 1: General statistics

2 Statistics of the DB

In this section, we will present some useful statistics of the database.

The original database has a total dimension of 14681.63 MB (?? $13.075GB$??), this encompasses both the physical size of the tables and the dimensions of primary and secondary indexes across various attributes.

In section A it can be found the complete statistics of the tables detailing its attributes along with the count of unique values, minimum, and maximum values for each attribute.

Here, to keep the focus on the four queries that we want to optimize, we will limit the presentation to some of that statistics. Specifically we decided to mention only the tables and the attributes that will be involved in at least one of the chosen query.

Attribute	Distinct values	Min value	Max value
c_custkey	1500000	1	1500000
c_name	1500000	'Customer#000000001'	'Customer#001500000'
c_address	1500000	-	-
c_nationkey	25	0	24
c_phone	1499963	-	-
c_acctbal	818834	'-999.99'	'9999.99'
c_comment	1496636	-	-

Table 2: Costumer statistics

Attribute	Distinct values	Min value	Max value
p_partkey	2000000	1	2000000
p_type	150	-	-
p_container	40	-	-

Table 3: Part statistics

Attribute	Distinct values	Min value	Max value
l_orderkey	15000000	1	60000000
l_partkey	2000000	1	2000000
l_quantity	50	1	50
l_extendedprice	1351462	900.91	104949.5
l_discount	11	'0.0'	'0.1'
l_tax	9	'0.0'	'0.08'
l_returnflag	3	-	-
l_linestatus	2	-	-
l_shipdate	2526	'1992-01-02'	'1998-12-01'
l_linenumber	7	1	7

Table 4: Lineitem statistics

Attribute	Distinct values	Min value	Max value
o_orderkey	15000000	1	60000000
o_custkey	999982	1	1499999
o_orderdate	2406	'1992-01-01'	'1998-08-02'

Table 5: Orders statistics

Attribute	Distinct values	Min value	Max value
n_nationkey	25	0	24
n_name	25	-	-

Table 6: Nation statistics

3 Query schemas

The assignment consists in using TPC-Benchmark H to test and optimize a set of four queries, using indexes, materialized views, a mixed approach of the two and fragmentation.

The set of queries selected for the assignment are Q1, Q10, Q14, Q17 of the Official Documentation. An overall view on the description and SQL implementation is given below.

3.1 Query 1

Brief description The Pricing Summary Report Query provides a summary pricing report for all lineitems shipped as of a given date. The date is within 60 - 120 days of the greatest ship date contained in the database. The query lists totals for extended price, discounted extended price, discounted extended price plus tax, average quantity, average extended price, and average discount. These aggregates are grouped by RETURNFLAG and LINESTATUS, and listed in ascending order of RETURNFLAG and LINESTATUS. A count of the number of lineitems in each group is included.

Functional definition

```
SELECT
    l_returnflag,
    l_linestatus,
    SUM(l_quantity) AS sum_qty,
    SUM(l_extendedprice) AS sum_base_price,
    SUM(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
    SUM(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    AVG(l_quantity) AS avg_qty,
    AVG(l_extendedprice) AS avg_price,
    AVG(l_discount) AS avg_disc,
    COUNT(*) AS count_order
FROM
    lineitem
WHERE
    l_shipdate <= DATE '1998-12-01' - INTERVAL '[DELTA]' DAY
GROUP BY
    l_returnflag,
    l_linestatus
ORDER BY
```

```
    l_returnflag,
    l_linestatus;
```

For a matter of simplicity we decided to take as slicing values the ones proposed by the validation paragraph in the official documentation. (So `'[DELTA]' = '90'`)

3.2 Query 10

Brief description The Returned Item Reporting Query finds the top 20 customers, in terms of their effect on lost revenue for a given quarter, who have returned parts. The query considers only parts that were ordered in the specified quarter. The query lists the customer's name, address, nation, phone number, account balance, comment information and revenue lost. The customers are listed in descending order of lost revenue. Revenue lost is defined as sum(l_extendedprice*(1-l_discount)) for all qualifying lineitems.

Functional definition

```
SELECT
    c_custkey,
    c_name,
    SUM(l_extendedprice * (1 - l_discount)) AS revenue,
    c_acctbal,
    n_name,
    c_address,
    c_phone,
    c_comment
FROM
    customer,
    orders,
    lineitem,
    nation
WHERE
    c_custkey = o_custkey
    AND l_orderkey = o_orderkey
    AND o_orderdate >= DATE '[DATE]'
    AND o_orderdate < DATE '[DATE]' + INTERVAL '3' MONTH
    AND l_returnflag = 'R'
    AND c_nationkey = n_nationkey
GROUP BY
    c_custkey,
    c_name,
    c_acctbal,
    c_phone,
    n_name,
    c_address,
    c_comment
ORDER BY
    revenue DESC;
```

For a matter of simplicity we decided to take as slicing values the ones proposed by the validation paragraph in the official documentation. (So '[DATE]' = '1993-10-01')

3.3 Query 14

Brief description The Promotion Effect Query determines what percentage of the revenue in a given year and month was derived from promotional parts. The query considers only parts actually shipped in that month and gives the percentage. Revenue is defined as (l_extendedprice * (1-l_discount)).

Functional definition

```
SELECT
    100.00 * SUM (CASE WHEN p_type like 'PROMO%'
                  THEN l_extendedprice*(1-l_discount)
                  ELSE 0 END) / SUM(l_extendedprice * (1 - l_discount))
    AS promo_revenue
FROM
    lineitem,
    part
WHERE
    l_partkey = p_partkey
    AND l_shipdate >= date '[DATE]'
    AND l_shipdate < date '[DATE]' + interval '1' month;
```

For a matter of simplicity we decided to take as slicing values the ones proposed by the validation paragraph in the official documentation. (So '[DATE]' = '1995-09-01')

3.4 Query 17

Brief description The Small-Quantity-Order Revenue Query considers parts of a given brand and with a given container type and determines the average lineitem quantity of such parts ordered for all orders (past and pending) in the 7-year database. What would be the average yearly gross (undiscounted) loss in revenue if orders for these parts with a quantity of less than 20 % of this average were no longer taken?

Functional definition

```
SELECT
    SUM(l_extendedprice) / 7.0 AS avg_yearly
FROM
    lineitem,
    part
WHERE
    p_partkey = l_partkey
    AND p_brand = '[BRAND]'
    AND p_container = '[CONTAINER]'
    AND l_quantity < (
```

```
                    SELECT
                        0.2 * AVG(l_quantity)
                    FROM
                        lineitem
                    WHERE
                        l_partkey = p_partkey
                );
```

For a matter of simplicity we decided to take as slicing values the ones proposed by the validation paragraph in the official documentation. (So `'[BRAND]' = 'Brand#23'` and `'[CONTAINER]' = 'MED BOX'`)

4 Baseline

We decided to test the execution time of queries without additional indexes or views, other than the keys suggested by the documentation, and to use it as a baseline for further improvement in the management of the queries.

We obtained the execution times using the *EXPLAIN ANALYZE* function available in PostgreSQL. The times represent the total estimated cost to execute the (entire) query. They are the sum of the startup cost and the cost to process all rows.

Every query has been tested five times, in order to record the mean and the standard deviation of the execution time, leading to more robust results.

The results of the tests are reported in the plot at table 7. [ADD COMMENT ON THE QUERY 17 RESULTS]

Query	Mean	Std
Q. 1	41.778	1.412
Q. 10	33.077	1.634
Q. 14	28.253	1.239
Q. 17	N/A	N/A

Table 7: Execution times of query

5 Indexes

Our first attempt was to add indexes on foreign keys, but in almost all cases, they weren't used, or didn't bring too many advantages, compared with their size.

For the sake of completeness we will report anyway the results (???).

Our second attempt was to add indexes on the attributes used for slicing, so involved in the `WHERE` condition.

Table	Attribute	Used in Query	Creation time [s]	Index size [MB]
lineitem	l_shipdate	Q1	32.43	397.54
lineitem	l_returnflag	Q1	61.83	396.46
lineitem	l_partkey	Q17	46.99	429.50
orders	o_orderdate	Q10	8.10	

Table 8: Indexes dimensions

With these indexes, which are ensured to be used in the execution of the queries, resulted in a total database size of $20.55GB$.

The execution time of the queries is summarized in the table below.

Query	Mean	Std
Q1	30.009	1.224
Q10	25.677	1.799
Q14	23.607	0.327
Q17	11.232	0.967

Table 9: Execution times of query with indexes

6 Materialized views

In this section we will propose some materialized views that aim to improve the execution time of the chosen queries.

6.1 Lineitem-part

In order to improve performances in executing query 14 we decided to create a materialized view as follow:

```
CREATE MATERIALIZED VIEW part_lineitem AS
SELECT
    l_returnflag,
    l_linestatus,
    l_quantity,
    l_extendedprice,
    l_discount,
    l_tax,
    l_shipdate,
    l_partkey,
    p_partkey,
    p_brand,
    p_container,
    SUBSTRING(p_type FROM 1 FOR 5) AS p_type_prefix,
    0.2 * AVG(l_quantity) OVER (PARTITION BY l_partkey) AS avg_quantity
```

```
FROM
    lineitem l
JOIN
    part p ON l.l_partkey = p.p_partkey;
```

Materializing the join operation of query 14 we expect to have a lower execution time with respect to the baseline.

Statistics of this view:

- Required time to create the view: 299.809 seconds
- Size of the view: 6.43 GB.

6.2 Costumer-orders-lineitem-nation

7 Mixed approach

8 Fragmentation

We considered to implement the fragmentation only for the tables of lineitem and orders since they are the most computationally expansive to scan entirely and since they are strictly involved in the set of our chosen queries.

While designing the fragmentation, we have always considered the broader aspect of the database as a decision-making tool, avoiding introducing too specific partitions that could have improved the specific set of chosen queries, but would have unnecessarily burdened the database as they were not generalizable to other queries. Therefore, we decided to consider a temporal fragmentation, as the temporal dimension is often involved in slicing conditions, even outside of the chosen queries.

In particular we fragmented the orders table with respect to the **o_orderdate** attribute. Each partitioned table contains a time-span of three months to allow a significant improvement in executing query 10. Furthermore we introduced in the partitioned tables a primary key in o_orderkey and a foreign key on o_custkey referencing c_custkey.

For the lineitem table we decided to use a partition on **l_shipdate** and a sub-partition on **l_returnflag**, to allow exploiting this partitioning in query 1, 10, 14.

The results are reported in the table 10

Query	Mean	Std
Q. 1	60.109	4.050
Q. 10	19.369	4.091
Q. 14	3.834	0.160
Q. 17	311.702	85.586

Table 10: Execution times of query with fragmented db

9 Conclusions

A Appendix A

Complete statistics of the database.

Attribute	Distinct values	Min value	Max value
c_custkey	1500000	1	1500000
c_name	1500000	'Customer#000000001'	'Customer#001500000'
c_address	1500000	-	-
c_nationkey	25	0	24
c_phone	1499963	-	-
c_acctbal	818834	'-999.99'	'9999.99'
c_mktsegment	5	-	-
c_comment	1496636	-	-

Table 11: Costumer statistics

Attribute	Distinct values	Min value	Max value
s_suppkey	100000	1	100000
s_name	100000	'Supplier#000000001'	'Supplier#000100000'
s_address	100000	-	-
s_nationkey	25	0	24
s_phone	100000	-	-
s_acctbal	95588	'-999.92'	'9999.93'
s_comment	99983	-	-

Table 12: Supplier statistics

Attribute	Distinct values	Min value	Max value
p_partkey	2000000	1	2000000
p_name	1999828	-	-
p_mfgr	5	Manufacturer#1	Manufacturer#5
p_brand	25	Brand#11	Brand#55
p_type	150	-	-
p_size	50	1	50
p_container	40	-	-
p_retailprice	31681	900.91	2098.99
p_comment	806046	-	-

Table 13: Part statistics

Attribute	Distinct values	Min value	Max value
ps_partkey	2000000	1	2000000
ps_supplierkey	100000	1	100000
ps_availqty	9999	1	9999
ps_supplycost	99901	1.0	1000.0
ps_comment	7914164	-	-

Table 14: Partsupp statistics

Attribute	Distinct values	Min value	Max value
l_orderkey	15000000	1	60000000
l_partkey	2000000	1	2000000
l_supplierkey	100000	1	100000
l_linenumber	7	1	7
l_quantity	50	1	50
l_extendedprice	1351462	900.91	104949.5
l_discount	11	0.0	0.1
l_tax	9	0.0	0.08
l_returnflag	3	-	-
l_linestatus	2	-	-
l_shipdate	2526	'1992-01-02'	'1998-12-01'
l_commitdate	2466	'1992-01-31'	'1998-10-31'
l_receiptdate	2555	'1992-01-03'	'1998-12-31'
l_shipinstruction	4	-	-
l_shipmode	7	-	-
l_comment	34378943	-	-

Table 15: Lineitem statistics

Attribute	Distinct values	Min value	Max value
o_orderkey	15000000	1	60000000
o_custkey	999982	1	1499999
o_orderstatus	3	-	-
o_totalprice	11944103	838.05	558822.56
o_orderdate	2406	'1992-01-01'	'1998-08-02'
o_orderpriority	5	-	-

Table 16: Orders statistics

Attribute	Distinct values	Min value	Max value
r_regionkey	5	0	4
r_name	5	-	-
r_comment	5	-	-

Table 17: Region statistics

Attribute	Distinct values	Min value	Max value
n_nationkey	25	0	24
n_name	25	-	-
n_regionkey	25	0	4
n_comment	25	-	-

Table 18: Nation statistics