Assignment - Data Managment for Big Data

Isac Pasianotto & Davide Rossi

Table of contents

1 Introduction

1.1 Database description

The database coming from the TPC Benchmark H consists of 8 tables of different sizes and different number of attributes. The tables size is scalable, depending on a scale factor (SF) which for the project purposes has been fixed to 10. Each table has its primary key and one or more secondary keys. A comprehensive description of tables and other specifications of the database can be found here, while the SQL implementation of the creation can be found here. Following, we'll just see a summary of the previous information, with all the attributes which will be used inside the project, where the attributes in bold are effectively used inside the queries, and the others are foreign keys used for join constraints:

	Rows	Attributes	Primary key	Total size (including PK)
Lineitem	SF*6.000.000	16	l_orderkey, l_linenumber	9.83 GB
Orders	SF*1.500.000	9	o_orderkey	2.3 GB
Partsupp	SF*800.000	5	ps_partkey, ps_suppkey	1.5 GB
Part	SF*200.000	9	p_partkey	362.88 MB
Customer	SF*150.000	8	c_custkey	312.05 MB
Supplier	SF*10.000	7	s_suppkey	19.47 MB
Nation	25	4	n_nationkey	24 KB
Region	5	3	r_regionkey	24 KB

LINEITEM

Attribute	Distinct values	Min value	Max value
l_extendedprice	1.351.462	900.91	104.949.50
l_discount	11	0.00	0.10
l_returnflag	3	A	R
l_orderkey	15.000.000	1	60.000.000
l_partkey	2.000.000	1	2.000.000
l_suppkey	100.000	1	100.000

ORDERS

Attribute	Distinct values	Min value	Max value
o_orderdate	2406	1992-01-01	1998-08-02
o_orderkey	15.000.000	1	60.000.000
o_custkey	999982	1	1.499.999

PART

Attribute	Distinct values	Min value	Max value
p_type	150	ECONOMY ANODIZED BRASS	STANDARD POLISHED TIN
p_partkey	2.000.000	1	2.000.000

CUSTOMER

Attribute	Distinct values	Min value	Max value
c_name	1.500.000	Customer#000000001	Customer#001500000
c_custkey	1.500.000	1	1.500.000
c_nationkey	25	0	24

SUPPLIER

Attribute	Distinct values	Min value	Max value
s_suppkey	100.000	1	100.000
s_nationkey	25	0	24

NATION

Attribute	Distinct values	Min value	Max value
n_name	25	ALGERIA	VIETNAM
n_nationkey	25	0	24
n_regionkey	5	0	4

REGION

Attribute	Distinct values	Min value	Max value
r_name	5	AFRICA	MIDDLE EAST
r_regionkey	5	0	4

1.2 Exam Assignment

The project that was given to us consists in using the TPC Benchmark H to test and optimize some queries, using indexes and materialized views in order to improve the execution time and the overall performances. In particular, two queries are requested to be improved: one for the export/import revenue value, and the other chosen between the late delivery and the returned item loss, where we chose the second.

1.3 Implementation of the queries

The first query is the following:

“Aggregation of the export/import of revenue of lineitems between two different nations (E,I) where E is the nation of the lineitem supplier and I the nations of the lineitem customer (export means that the supplier is in the nation E and import means is in the nation I). The aggregations should be performed with the following roll-up:
Month - Quarter - Year
Type
Nation - Region
The slicing is over Type and Exporting nation.”

The query has been implemented in SQL in the following way:

```sql
-- Query 1
SELECT  (NS.n_name || ', ' || NC.n_name)        AS Nation,
        (RS.r_name || ', ' || RC.r_name)        AS Region,
        SUM(L.l_extendedprice*(1-L.l_discount)) AS revenue,
        DATE_PART('month', O.o_orderdate)       AS monthOrder,
        DATE_PART('quarter', O.o_orderdate)     AS quarterOrder,
        DATE_PART('year', O.o_orderdate)        AS yearOrder,
        P.p_type                                AS ptype
FROM  LINEITEM AS L                                   JOIN
      ORDERS   AS O  ON  L.l_orderkey=O.o_orderkey    JOIN
      PART     AS P  ON  P.p_partkey=L.l_partkey      JOIN
      SUPPLIER AS S  ON  S.s_suppkey=L.l_suppkey      JOIN
      CUSTOMER AS Cu ON Cu.c_custkey=O.o_custkey      JOIN
      NATION   AS NS ON NS.n_nationkey=S.s_nationkey  JOIN
      NATION   AS NC ON NC.n_nationkey=Cu.c_nationkey JOIN
      REGION   AS RS ON RS.r_regionkey=NS.n_regionkey JOIN
      REGION   AS RC ON RC.r_regionkey=NC.n_regionkey
WHERE P.p_type='ECONOMY ANODIZED TIN' AND
      NS.n_name='CHINA'
GROUP BY ROLLUP(ptype),
         ROLLUP(Region,Nation),
         ROLLUP(yearOrder,quarterOrder,monthOrder)
```

The third query is the following:

“The query gives the revenue loss for customers who might be having problems with the parts that are shipped to them.
The aggregations should be performed with the following roll-up
Month - Quarter - Year
Customer
The query can be issued with the following slicing (combined)
Name of a customer
A specific quarter”

The query has been implemented in SQL in the following way:

```sql
-- Query 3
SELECT    SUM(L.l_extendedprice*(1-L.l_discount))  AS revenue,
          DATE_PART('month', O.o_orderdate)        AS monthOrder,
          DATE_PART('quarter', O.o_orderdate)      AS quarterOrder,
          DATE_PART('year', O.o_orderdate)         AS yearOrder,
          CU.c_name                                AS custName
FROM  LINEITEM AS L                                JOIN
      ORDERS   AS O  ON  L.l_orderkey=O.o_orderkey JOIN
      CUSTOMER AS CU ON CU.c_custkey=O.o_custkey
WHERE L.l_returnflag='R'                      AND
      CU.c_name='Customer#000001999'          AND
      DATE_PART('quarter', O.o_orderdate)='1'
GROUP BY ROLLUP(yearOrder,quarterOrder,monthOrder),
         ROLLUP(custName)
```

1.4 Measurements techniques

In order to assess the performance of the queries we have measured their execution time. In particular this was done by saving each query in a dedicated SQL file and using the Unix command `time` to measure the execution of the `psql` command, which is used to run a sql file with Postgresql. In order to avoid time waste due to the creation of the output, we have run the queries with the option `EXPLAIN ANALYZE`, so that the output would have been limited to the solely query execution plan, and as further measure we have also redirected the output to `/dev/null` in order to avoid altering the measurements with the time needed to print the results on the terminal. An example of the described procedure is the following:

```
time psql -U $dbuser -d $dbname -f $fname.sql > /dev/null

real    0m9.584s
user    0m0.054s
sys     0m0.013s
```

Where `dbuser` is the username of the account used to access the database, `dbname` is the name of the database, and `fname` is the name of the file containing the query to be executed. The output of the `time` command is particularly useful, since it gives not just the "wall clock" time (the `real` time), but also the time spent by the CPU doing operations (which can be obtained by summing `user` and `sys` value). This is interesting since usually, in the context of query optimization, the most time consuming part is not the operations done by the CPU but the time spent waiting for the disk to read/write data. The results of $\texttt{real} - (\texttt{user} + \texttt{sys})$ can be considered as a good approximation of the time spent accessing the disk.

In order to have a statistically significative measurement we have run each query 75 times, and considered the average and the standard deviation of the results.

Also, to avoid the result to possibly depend on the particular choice of the slicing, we have run each query with 3 different slicing values:

```
-- Query 1

-- Case: a
-- [...]
WHERE P.p_type='ECONOMY ANODIZED TIN' AND NS.n_name='CHINA'
-- [...]

-- Case: b
-- [...]
WHERE P.p_type='PROMO BURNISHED TIN' AND NS.n_name='CANADA'
-- [...]
```

```sql
-- Case: c
-- [...]
WHERE P.p_type='LARGE BRUSHED BRASS' AND NS.n_name='MOZAMBIQUE'
--[...]
```

```sql
-- Query 3

-- Case: a
-- [...]
WHERE L.l_returnflag='R' AND CU.c_name='Customer#000001999'
      AND DATE_PART('quarter', O.o_orderdate)='1'
-- [...]

-- Case: b
-- [...]
WHERE L.l_returnflag='R' AND CU.c_name='Customer#000074236'
      AND DATE_PART('quarter', O.o_orderdate)='2'
-- [...]

-- Case: c
-- [...]
WHERE L.l_returnflag='R' AND CU.c_name='Customer#000002345'
      AND DATE_PART('quarter', O.o_orderdate)='4'
-- [...]
```

Regarding the size of the whole database, we have checked the value reported in the *Statistics* tab of the properties of the database in pgAdmin4 every time the structure of the database was altered.

1.5 Hardware

The project has been run on a laptop with the following specifics:

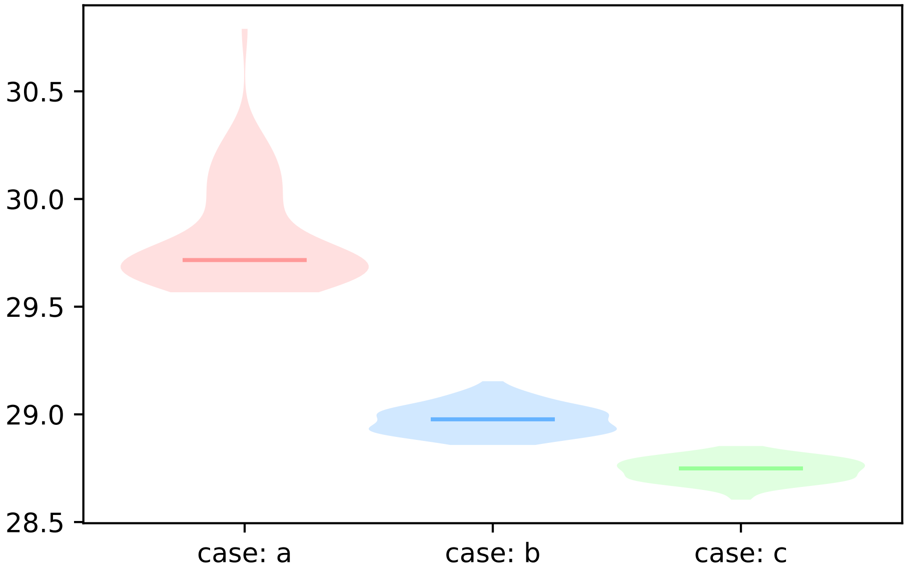
- CPU: Intel Core i7-5500U @ 2.40GHz, 2 cores, 4 threads
- RAM: 8 GB DDR3, 1600 MT/s
- SSD SATA Kingston 240 GB
- GPU: NVIDIA GeForce 940M
- OS: Pop!_OS 22.04 LTS x86_64

2 Baseline: database with no optimization

The first thing we have tested is the execution time of the queries on the database without any optimization. In order to do this we run the previous queries without any index or materialized view. The size of the database in this case is 14.33 GB. The results of the measurements are reported in the following tables:

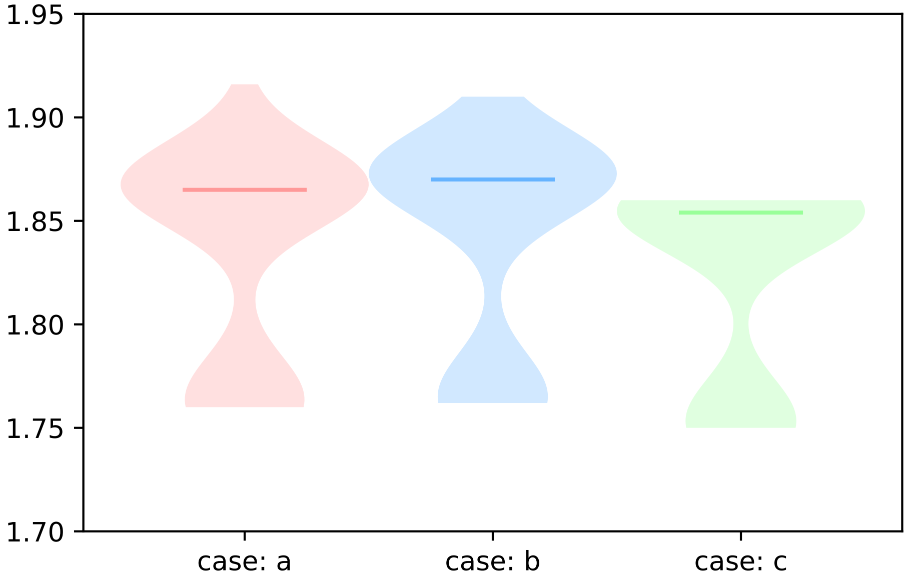
Query 1, baseline: no optimization

Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	29.716500	29.825792	0.234098	0.062306
b	28.977000	28.976956	0.068102	0.063265
c	28.749000	28.742418	0.053242	0.063716



Query 3, baseline: no optimization

Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	1.865000	1.835216	0.049357	0.035000
b	1.870000	1.840905	0.049831	0.035149
c	1.854000	1.823243	0.046897	0.034932



3 Indexes

3.1 Foreign Keys

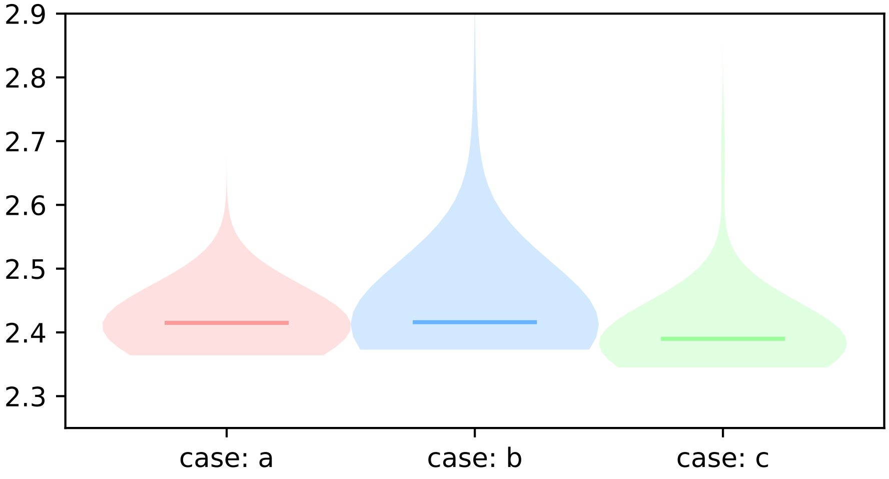
The next step was to create indexes on attributes. We have created an index on each foreign key used for queries, then we have run the queries and analyzed the query execution plan in order to understand which indexes the optimizer decided to use, with the following results:

Attribute	Table	Used?	Weight if used
l_partkey	LINEITEM	YES (Q1)	429.51 MB
s_nationkey	SUPPLIER	YES (Q1)	704 KB
o_custkey	ORDERS	YES (Q3)	58.12 MB
l_orderkey	LINEITEM	YES (Q3)	120.24 MB
l_suppkey	LINEITEM	NO	
c_nationkey	CUSTOMER	NO	
n_regionkey	NATION	NO	

We have then removed the not used ones and measured again the execution time. The total space used by the used indexes is 608.57 MB. The results are reported in the following tables:

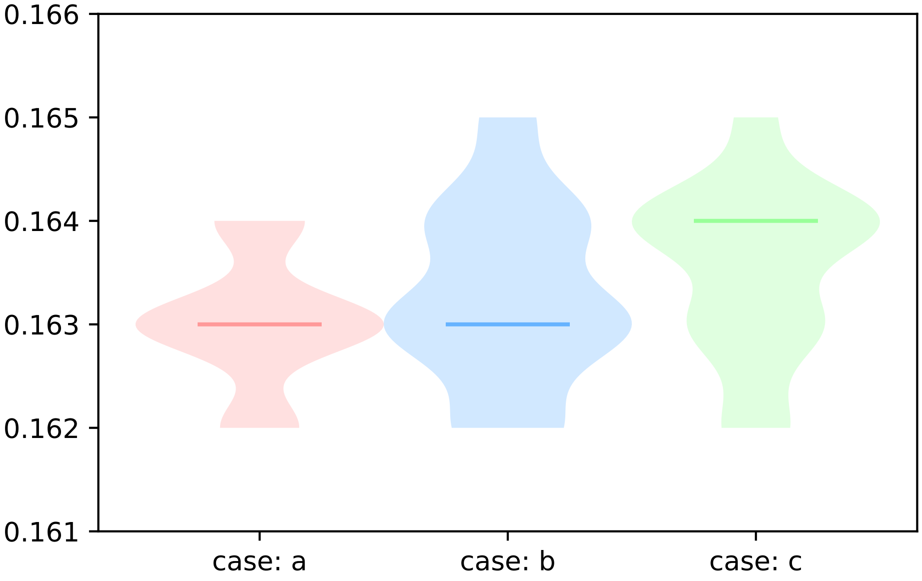
Query 1, indexes on foreign keys

Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	2.415000	2.426189	0.142716	0.061905
b	2.416000	2.444205	0.225742	0.063014
c	2.390000	2.404123	0.141963	0.063452



Query 3, indexes on foreign keys

Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	0.163000	0.163027	0.000636	0.045973
b	0.163000	0.163284	0.000878	0.045905
c	0.164000	0.163534	0.000829	0.046055



3.2 Other attributes

In order to further improve the performance of the queries we have created indexes on other attributes that were likely to be used in the queries (for example, the ones used in where clauses). Then we have run the queries again and analyzed the query execution plan, with the following results:

List of other attributes used in the queries

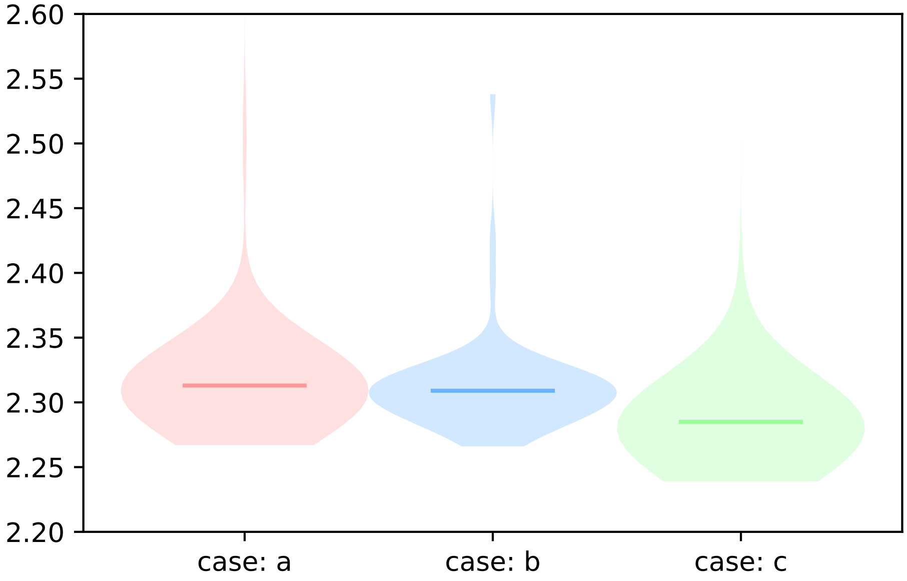
Attribute	Table	Used?	Weight if used
p_type	PART	YES (Q1)	13.66 MB
c_name	CUSTOMER	YES (Q3)	58.12 MB
n_name	NATION	NO	
r_name	REGION	NO	
l_returnflag	LINEITEM	NO	

Attribute	Table	Used?	Weight if used
o_orderdate	ORDERS	NO	
l_extendedprice	LINEITEM	NO	
l_discount	LINEITEM	NO	

We have then removed the not used ones and measured again the execution time. The total space used by the used indexes is 71.78 MB. The results are reported in the following tables:

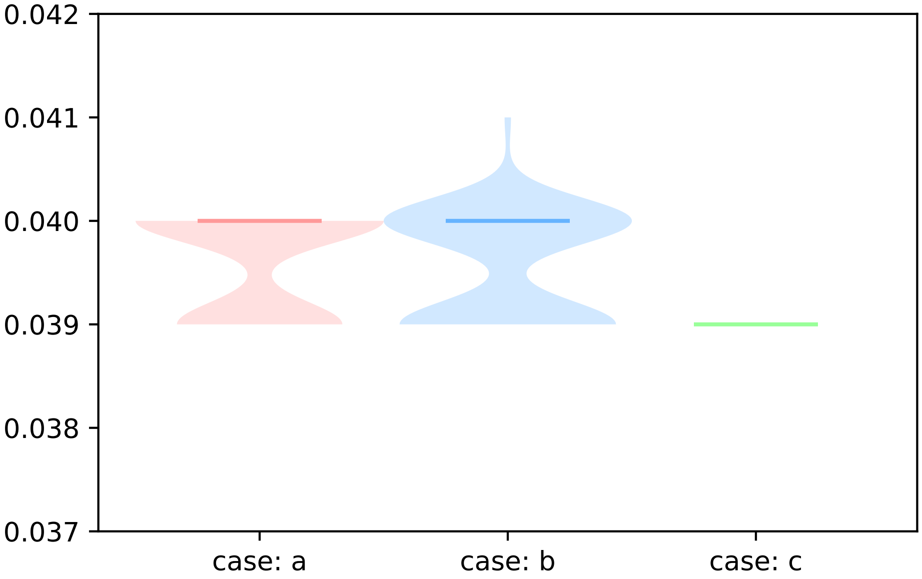
Query 1, other indexes

Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	2.313000	2.320342	0.080370	0.047822
b	2.309000	2.311042	0.036061	0.047347
c	2.285000	2.290959	0.088198	0.047890



Query 3, other indexes

Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	0.040000	0.039600	0.000490	0.034386
b	0.040000	0.039554	0.000524	0.034284
c	0.039000	0.039000	0.000000	0.033613



4 Materialized View

As third optimization step, we tried to create a materialized view. In particular, we have tested two different views to understand if the use of a "smaller" or a "bigger" view could improve the performance of the queries.

4.1 Big materialized view

The first attempt we made is a "big" view, obtained by joining all the tables needed for the first query. Its size is 6.320 GB.

4.1.1 Creation of the view

```
CREATE MATERIALIZED VIEW big_view AS
SELECT L.l_extendedprice, L.l_discount, O.o_orderdate, P.p_type,
S.s_nationkey, Cu.c_nationkey,Cu.c_name, L.l_returnflag
FROM  LINEITEM AS L JOIN
        ORDERS AS O  ON  L.l_orderkey=O.o_orderkey JOIN
          PART AS P  ON  P.p_partkey=L.l_partkey   JOIN
      SUPPLIER AS S  ON  S.s_suppkey=L.l_suppkey   JOIN
      CUSTOMER AS Cu ON Cu.c_custkey=O.o_custkey
```

4.1.2 Changes to the queries

In order to use the materialized view we have changed the queries in the following way:

- query 1:

```
SELECT (NS.n_name || ', ' || NC.n_name)        AS Nation,
       (RS.r_name || ', ' || RC.r_name)        AS Region,
       SUM(V.l_extendedprice*(1-V.l_discount)) AS revenue,
       DATE_PART('month', V.o_orderdate)       AS monthOrder,
       DATE_PART('quarter', V.o_orderdate)     AS quarterOrder,
       DATE_PART('year', V.o_orderdate)        AS yearOrder,
       V.p_type                                AS ptype
FROM big_view AS V JOIN
       NATION AS NS ON NS.n_nationkey=V.s_nationkey  JOIN
       NATION AS NC ON NC.n_nationkey=V.c_nationkey  JOIN
       REGION AS RS ON RS.r_regionkey=NS.n_regionkey JOIN
       REGION AS RC ON RC.r_regionkey=NC.n_regionkey
WHERE V.p_type='ECONOMY ANODIZED TIN' AND NS.n_name='CHINA'
GROUP BY ROLLUP(ptype),
         ROLLUP(Region,Nation),
         ROLLUP(yearOrder,quarterOrder,monthOrder)
```

- query 3:

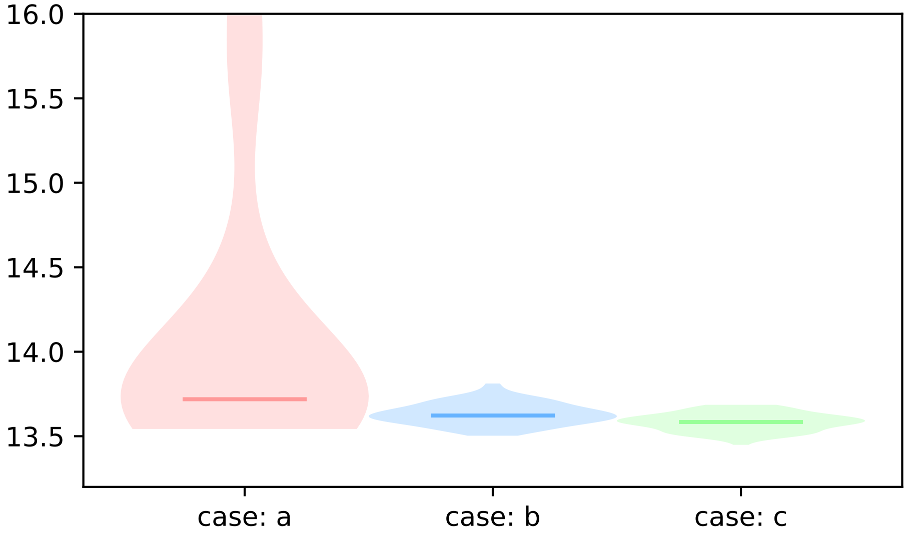
```
SELECT SUM(l_extendedprice*(1-l_discount)) AS revenue,
     DATE_PART('month', o_orderdate)      AS monthOrder,
     DATE_PART('quarter', o_orderdate)    AS quarterOrder,
     DATE_PART('year', o_orderdate)       AS yearOrder,
     c_name                               AS custName
FROM big_view
WHERE L.l_returnflag='R' AND CU.c_name='Customer#000001999' AND
       DATE_PART('quarter', O.o_orderdate) = '1'
GROUP BY ROLLUP(yearOrder,quarterOrder,monthOrder),
         ROLLUP(custName);
```

4.1.3 Measurements

By running the queries with the big materialized view we have obtained the following results:

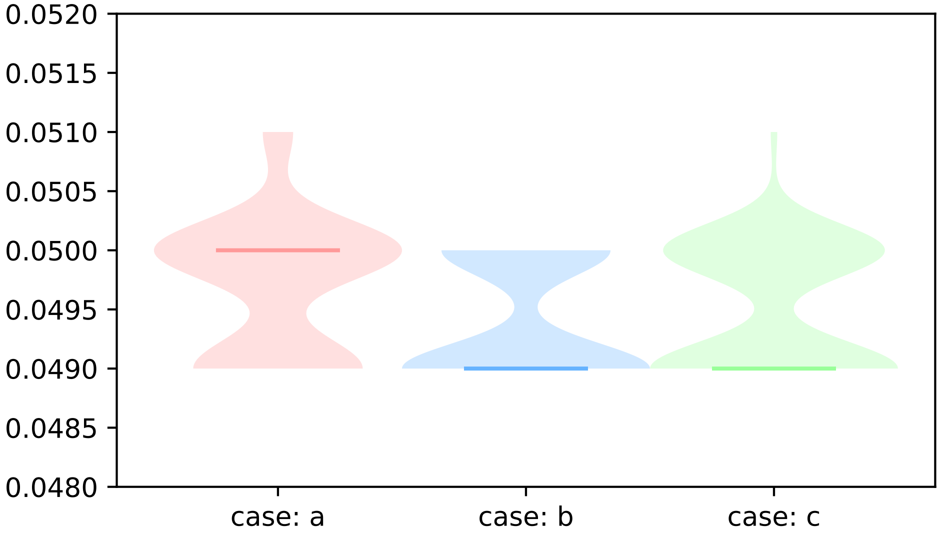
Query 1

Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	13.719000	14.204466	0.927408	0.047849
b	13.622500	13.631618	0.064961	0.047162
c	13.584000	13.583563	0.053934	0.047169



Query 3

Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	0.050000	0.049689	0.000591	0.045784
b	0.049000	0.049405	0.000491	0.045649
c	0.049000	0.049493	0.000527	0.045712



4.2 Small materialized view

The second attempt we made is a “small” view, obtained by joining all the tables needed for the second query. Its size is 5.066 GB.

4.2.1 Creation of the view

```
CREATE MATERIALIZED VIEW small_view AS
SELECT L.l_extendedprice, L.l_discount, O.o_orderdate, Cu.c_nationkey,
Cu.c_name, L.l_returnflag, L.l_suppkey, L.l_partkey
FROM LINEITEM AS L JOIN
       ORDERS AS O  ON  L.l_orderkey=O.o_orderkey JOIN
     CUSTOMER AS Cu ON Cu.c_custkey=O.o_custkey;
```

4.2.2 Changes to the queries

In order to use the materialized view we have changed the queries in the following way:

- query 1:

```
SELECT (NS.n_name || ', ' || NC.n_name)        AS Nation,
       (RS.r_name || ', ' || RC.r_name)        AS Region,
       SUM(V.l_extendedprice*(1-V.l_discount)) AS revenue,
       DATE_PART('month', V.o_orderdate)       AS monthOrder,
       DATE_PART('quarter', V.o_orderdate)     AS quarterOrder,
       DATE_PART('year', V.o_orderdate)        AS yearOrder,
       V.p_type                                AS ptype
FROM small_view AS V JOIN
         NATION AS NS ON NS.n_nationkey=V.c_nationkey  JOIN
         NATION AS NC ON NC.n_nationkey=V.c_nationkey  JOIN
         REGION AS RS ON RS.r_regionkey=NS.n_regionkey JOIN
         REGION AS RC ON RC.r_regionkey=NC.n_regionkey JOIN
       SUPPLIER AS S  ON  S.s_suppkey=V.l_suppkey      JOIN
           PART AS P  ON  P.p_partkey=V.l_partkey
WHERE P.p_type='ECONOMY ANODIZED TIN' AND NS.n_name='CHINA'
GROUP BY ROLLUP(ptype),
         ROLLUP(Region,Nation),
         ROLLUP(yearOrder,quarterOrder,monthOrder);
```

- query 3:

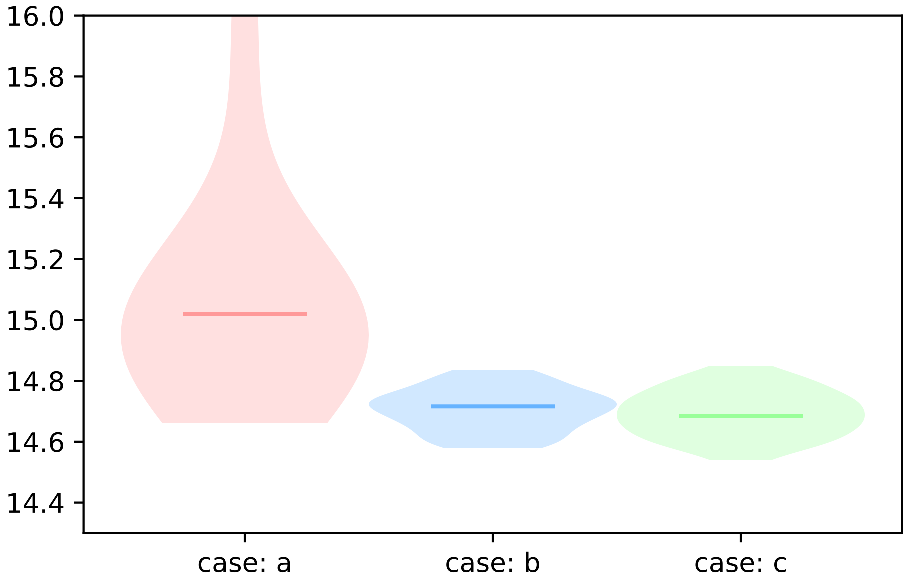
```
SELECT SUM(l_extendedprice*(1-l_discount)) AS revenue,
     DATE_PART('month', o_orderdate)     AS monthOrder,
     DATE_PART('quarter', o_orderdate)   AS quarterOrder,
     DATE_PART('year', o_orderdate)      AS yearOrder,
       c_name                            AS custName
FROM small_view
WHERE L.l_returnflag='R' AND CU.c_name='Customer#000001999' AND
       DATE_PART('quarter', O.o_orderdate) = '1'
GROUP BY ROLLUP(yearOrder,quarterOrder,monthOrder),
         ROLLUP(custName);
```

4.2.3 Measurements

By running the queries with the small materialized view we have obtained the following results:

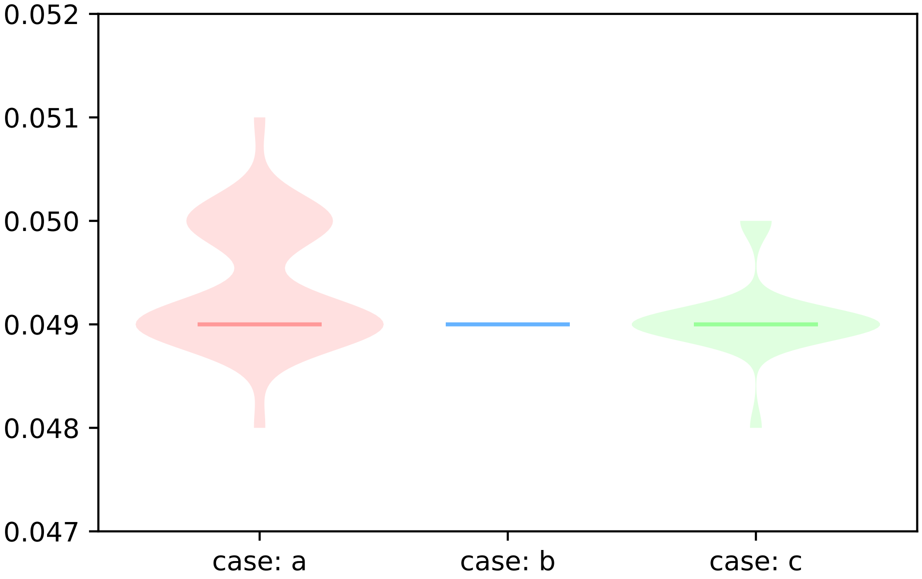
Query 1

Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	15.019000	15.252671	0.660407	0.037370
b	14.716000	14.706184	0.070695	0.037429
c	14.684000	14.692877	0.077298	0.037684



Query 3

Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	0.049000	0.049378	0.000586	0.045689
b	0.049000	0.049000	0.000000	0.045359
c	0.049000	0.049068	0.000380	0.045392



5 Materialized view with indexes

In order to further improve the performance of the queries we have created indexes on the materialized views. As done before, we have created indexes on all the attributes of the materialized views, we have run the queries and checked the query execution plan and we have removed the indexes that were not used.

5.1 Big materialized view

5.1.1 Index selection

The results of the index selection are reported in the following table:

List of attributes in the big materialized view

Attribute	Used?	Weight if used
p_type	YES (Q1)	407.15 MB
s_nationkey	YES (Q1)	396.47 MB
c_name	YES (Q3)	419.69 MB
l_extendedprice	NO	
l_discount	NO	
o_orderdate	NO	
c_nationkey	NO	
l_returnflag	NO	

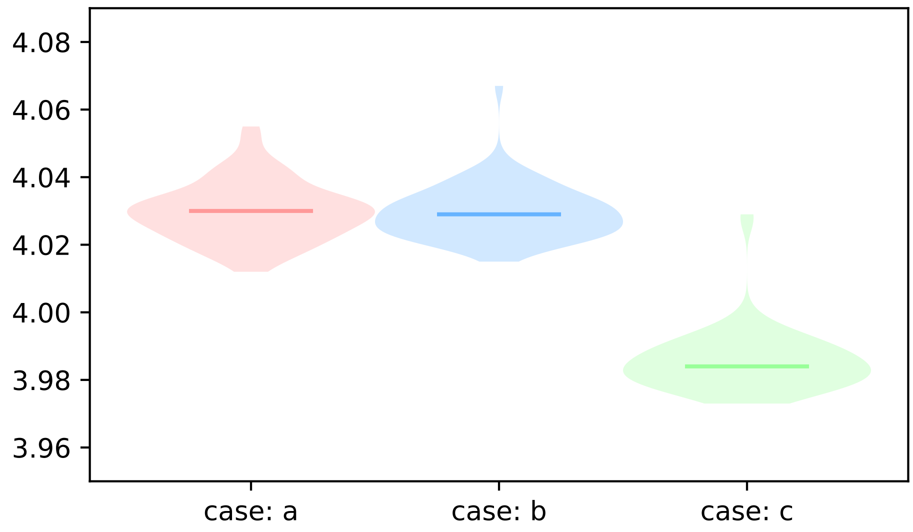
The total space used by the used indexes is 1.223 GB.

5.1.2 Measurements

The results are reported in the following tables:

Query 1

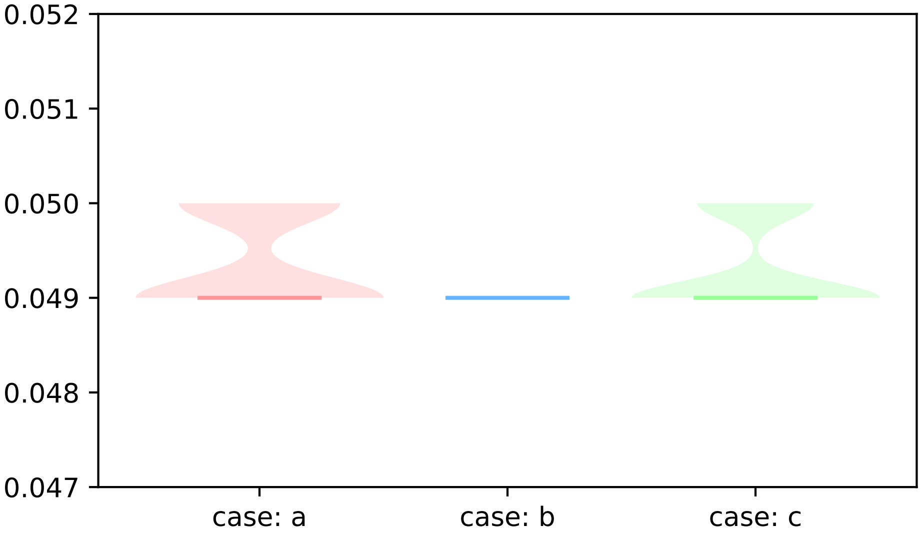
Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	4.030000	4.029797	0.008645	0.047189
b	4.029000	4.029458	0.007888	0.046972
c	3.984000	3.985514	0.009302	0.047125



Query 3, big materialized view with indexes

Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	0.049000	0.049394	0.000489	0.045549
b	0.049000	0.049000	0.000000	0.045375
c	0.049000	0.049319	0.000466	0.045625

Query 3, big materialized view with indexes



5.2 Small materialized view

5.2.1 Index selection

The results of the index selection are reported in the following table:

List of attributes in the small materialized view

Attribute	Used?	Weight if used
l_partkey	YES (Q1)	429.51 MB
c_nationkey	NO	
c_name	NO	
l_extendedprice	NO	
l_discount	NO	
o_orderdate	NO	
l_returnflag	NO	
l_suppkey	NO	

Note: query 1 executed with the small view also uses the index for the attribute `p_type` from the table `PART` (whose weight is 13.66 MB).

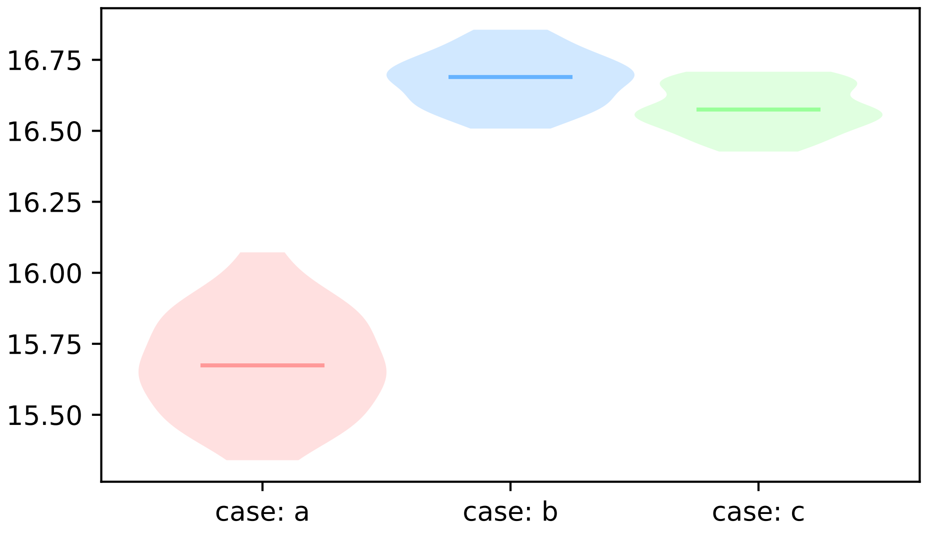
The total space used by the used indexes is 443.17 MB.

5.2.2 Measurements

The results are reported in the following tables:

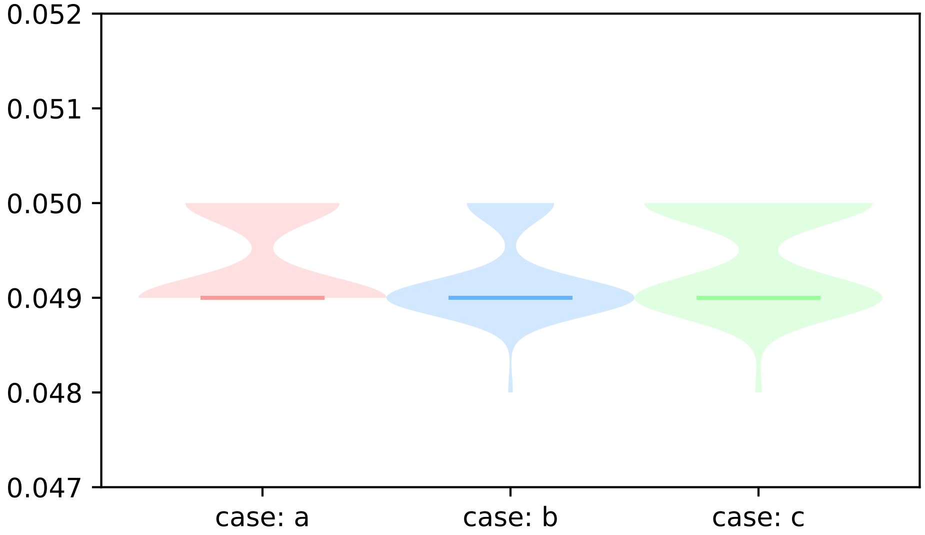
Query 1, small materialized view with indexes

Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	15.674000	15.680492	0.180330	0.037356
b	16.689500	16.679466	0.092544	0.037517
c	16.575000	16.580588	0.081720	0.037706



Query 3, small materialized view with indexes

Case	Real time - median	Real time - mean	Real time - SD	CPU time - mean
a	0.049000	0.049384	0.000486	0.045795
b	0.049000	0.049243	0.000459	0.045568
c	0.049000	0.049459	0.000525	0.045824



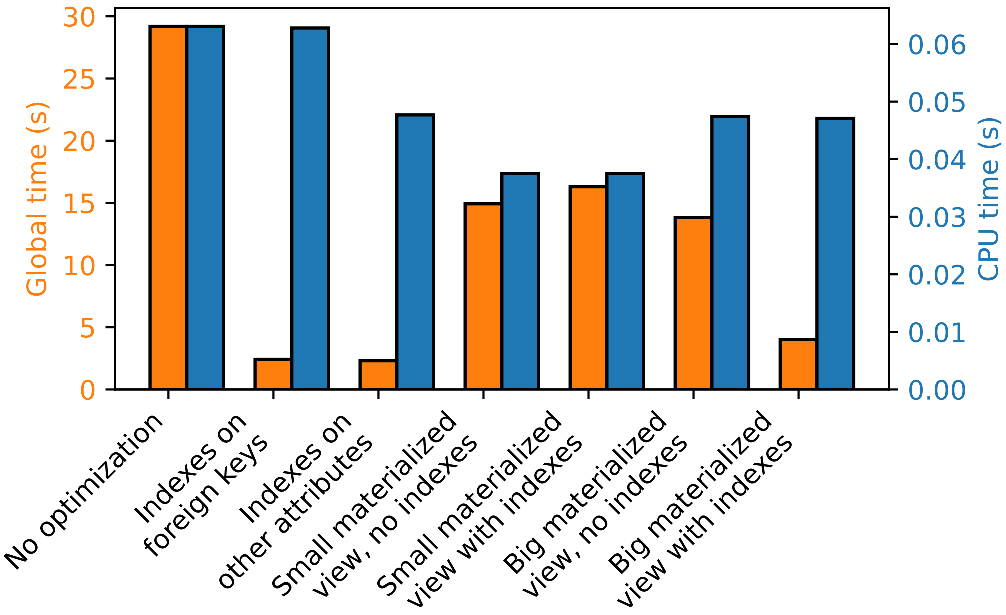
6 Final results

In the following tables we report the results of the measurements for each query, with the different optimizations we have tried. Since, as from the previous results, the choice for the slicing values basically does not impact on the execution time, for simplicity we decided to take the mean and median of all values, regardless the slicing value. We have also reported the weight of the database for each case, in order to understand the impact of the different optimizations on the database size:

Results of the measurements for query 1

Situation	median	mean	SD	cpu_mean	DB_weight(GB)
no optimization	28.987000	29.196290	0.491748	0.063077	14.330000
indexes on foreign keys	2.404000	2.424845	0.175257	0.062786	14.940000
other indexes	2.299000	2.307431	0.073132	0.047688	15.010000
small view	14.759000	14.924816	0.505162	0.037486	19.400000
small view with indexes	16.553500	16.298619	0.474129	0.037518	19.840000
big view	13.636500	13.812778	0.615987	0.047401	20.650000
big view with indexes	4.024000	4.015060	0.022472	0.047096	21.870000

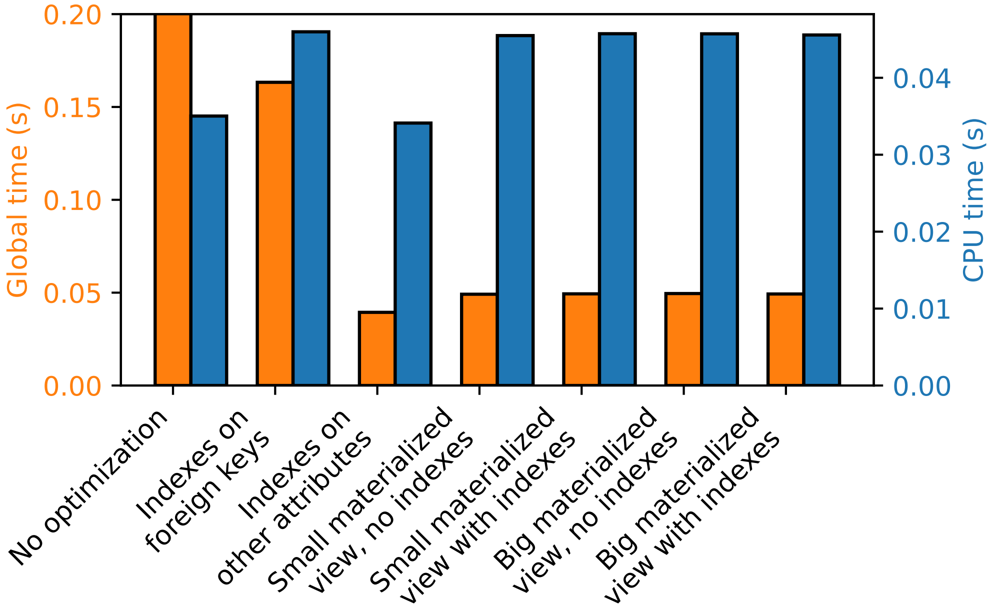
Summary of the results for query 1

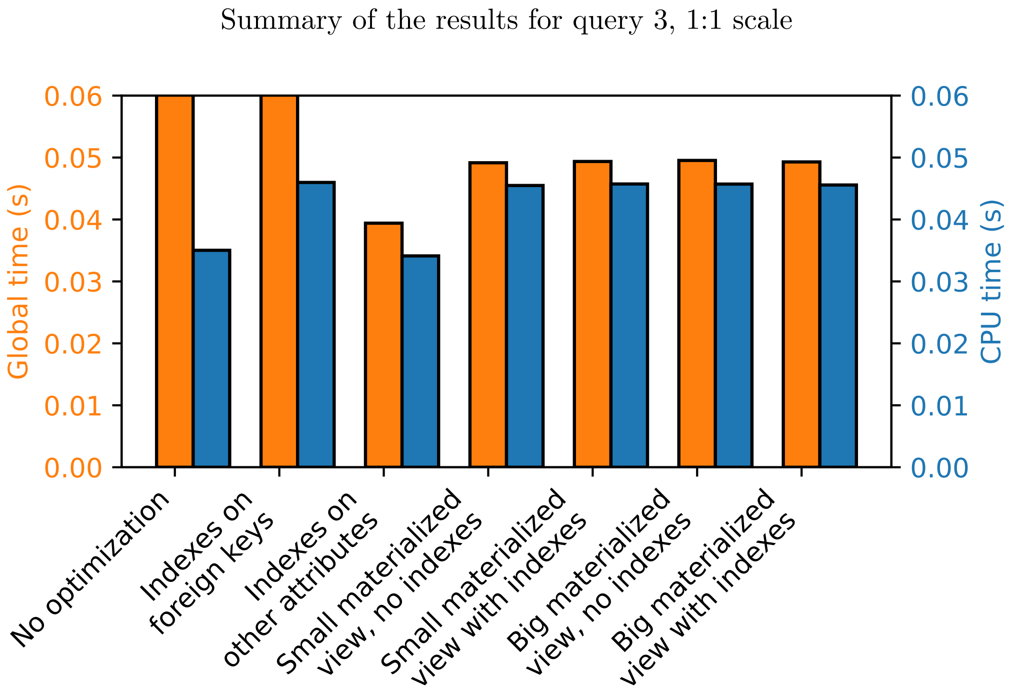


Results of the measurements for query 3

Situation	median	mean	SD	cpu_mean	DB_weight(GB)
no optimization	1.857000	1.833122	0.049265	0.035027	14.330000
indexes on foreign keys	0.163000	0.163281	0.000814	0.045977	14.940000
other indexes	0.039000	0.039403	0.000500	0.034117	15.010000
small view	0.049000	0.049156	0.000444	0.045486	19.400000
small view with indexes	0.049000	0.049362	0.000499	0.045729	19.840000
big view	0.050000	0.049529	0.000551	0.045715	20.650000
big view with indexes	0.049000	0.049290	0.000454	0.045565	21.870000

Summary of the results for query 3

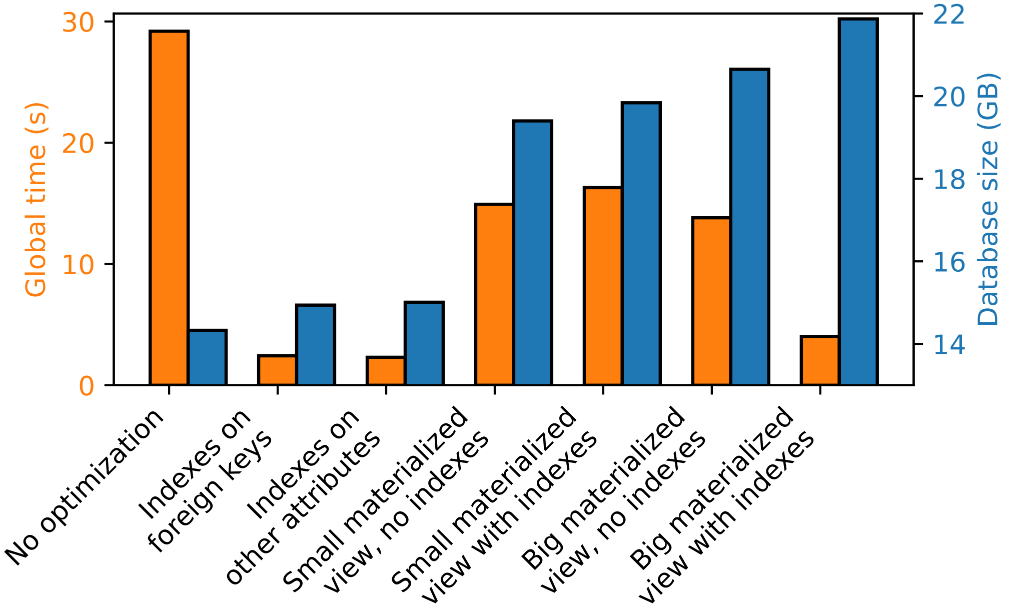



Summary of the results for query 3, 1:1 scale

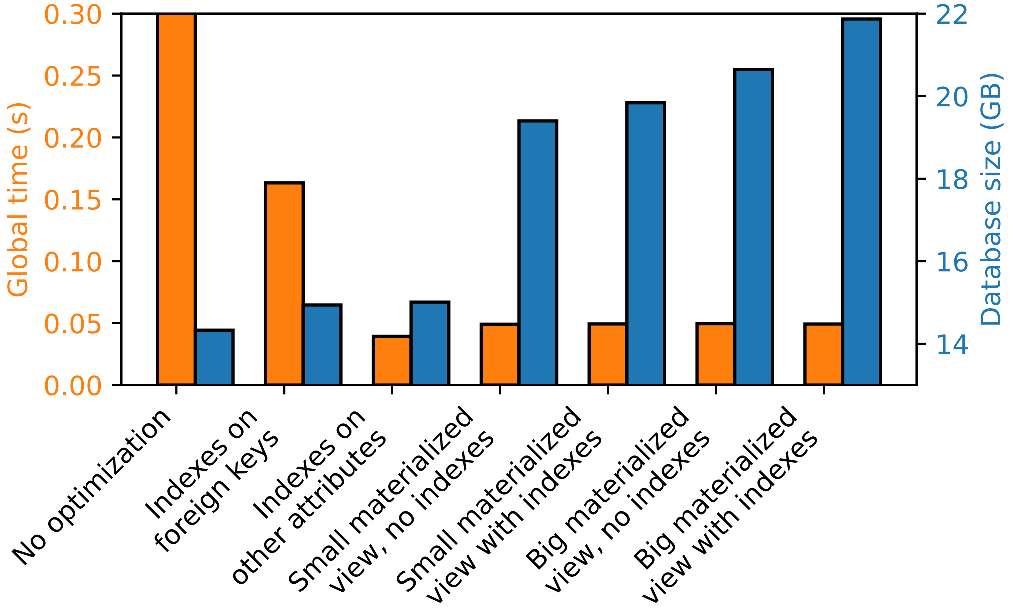
As from the previous graphs, the CPU time appears to be negligible with respect to the real time in the case of query 1, while representing a significant part of the total time in the case of query 3. This is due to the fact that query 1 is mainly I/O bound (since the output size is quite big, having more than 7000 lines), while query 3 is mainly CPU bound (since the output size is quite small, having only a few tenths of lines).

Let's also have a look at the trade-off between the size of the database and the performances. In the following graphs, we plot the size of the database (in GB) and the global time (in seconds) for each situation:

Query 1: trade-off size and performance



Query 3: trade-off size and performance



Given the previous results, the best compromise between performance and size of the database, which actually also gives the best overall performances for both queries, is given by the situation with indexes on foreign keys and other attributes.