Data Warehousing and Data Mining Project

Free University of Bolzano
Faculty of Computer Science

Ondrej Platek and Peteris Nikiforovs

Semester project

Lecturer: M.Sc. Johann Gamper, Ph.D.
Exerciser: M.Sc. Mateusz Pawlik

Autumn semester 2011

Evaluation: Every milestone is graded on a scale of 0 - 30. 25 points is the expected minimum. Project will be evaluated as average of all milestones. Some points will be added/subtracted from the average after the final presentation.

Contents

1. Milestone - Data warehouse design

1.1 Content of Milestone 1

This chapter contains report from the 1^{st} Milestone which purpose was to design a data warehouse from a chosen domain. We divided the tasks into several sections which we now introduce.

The Section 1.2 describes a business domain of a company which we have chosen for our sample data warehouse. The Subsections 1.2.1, 1.2.2 and 1.2.3 will introduce step by step the data warehouse objectives, the business processes and finally the granularity of the processes. The objectives will tell you why we need the data warehouse. The business processes will present you the core of the organisation and last but not least the granularity of the processes show you how detail information we want to capture.

The next Section 1.3 concentrates on the conceptual modeling. In Subsections 1.3.1 we define our facts. In Subsections 1.3.2 and 1.3.3 the measures, attributes and their hierarchies are depicted. In these Subsections the aggregation of measures is introduced and we remind the process granularity which we apply on our facts.

In the Section 1.4 we will depict two alternatives in logical design. Concretely, in Subsection 1.4.1 is presented star schema of our facts and in Subsection 1.4.2 we show snowflake schema. Finally, in Subsection 1.4.3 we argue why we have chosen the star schema.

The last Section 1.5 describes the data and the scripts we used to populate our database.

1.2 Business Domain

The domain of this data warehouse project is a fictional daily newspaper, modeled after The Wall Street Journal (WSJ or the Journal). The Journal was founded in the 19th century, was active throughout the 20th century as printed newspaper and in 1996 launched an online version.

It produces a daily printed edition which is delivered to subscribed customers worldwide and sold at newspaper stands. The same content is also available on the website. The online edition is available to authenticated paid subscribers only.

1.2.1 Data Warehouse Objectives

Our company has had multiple data sources (paper receipts, paper journals, text files, proprietary binary file databases and lately SQL databases) for the print newspaper over the years which have been integrated into one relational database. The online edition was designed separately and has its own databases.

Both databases store customer data differently which makes it hard to aggregate statistics from both editions. Also, the database for the printed edition was built additively and contains data inconsistencies. The data warehouse would allow to combine personalized knowledge of the online edition readers with a lot of additional historical data from the printed edition.

WSJ is the largest newspaper in the United States by circulation and has more than 400,000 online paid subscriptions. As such, the amount of data accumulated over the years is huge and was not designed to be stored for efficient analysis. As a result, it's very inefficient to extract useful information as the business queries would take a long time when run on production databases.

The management of Journal understands, that the Web not only is a thread to printed newspaper, but also could be a huge market and source of information. As goal for new decade the Journal company is interested among others in following topics:

- How to provide to advertisers guaranties and statistics that our readers will see their advertisement? (Consequently The Journal would be able to require more money from advertiser).
- How to arrange articles in the week in order to limit the amount of low read articles?
- Which articles put together in order to keep the reader reading?
- Find matches of convenient advertisement and articles based on pro click rate.
- Which advertisers we consider unimportant according our database, but have really successful campaigns and profit from our newspaper?
- When should we advertise the Journal to subscribers?

The WSJ decided to start building data warehouse based on the database containing new online data. Secondly, the WSJ woul like to add the historical data to the data warehouse. This scenario allows early result for current relevant data and it also allows incorporating historical data step by step in future. Further on, we work only with the data collected in the online database of the WSJ.

1.2.2 Business Processes

We have identified three main business processes that our newspaper cares about.

- Selling subscriptions
- Advertising
- Popularity of articles content

.

The Journal mainly generates revenue from subscription sales. Since it is an international newspaper, it's important to break down the sales by countries and regions. It's also necessary for the company shareholders to visualize the growth over the years, taking into account the data from both editions.

In addition to subscriptions, the Journal places ads in the paper and generates revenue from it. The company would like to know how much revenue it generates and compare it to the subscription sales.

Lastly, the paper cares about the quality of its content. The Journal mainly covers economics and business topics and financial news. Since it's possible to track all user actions on the website, the data warehouse would allow testing of introduction of new content, identifying most popular articles, determine least successful authors, etc..

1.2.3 Process Granularity

Selling subscriptions

We break down the subscriptions by location and date in order to see how successful we are during the time on different places. It is worth to mention that subscriptions are the main source of our income.

Highest granularity for location should be the **city**[1] , because it is the ideal unit for describing mass behaviour. The habits and traditions which influences the mass behaviour are best captured on the level of cities.

Since it's a global newspaper, countries should be grouped by regions, continents and arbitrary zones based on detail information our newspaper has about the specified region.

It's not important to know at exactly what time a subscription was purchased, but the date is relevant. In order to differentiate holidays, day of week we have chosen **date** as our highest granularity.

Advertising

The process of advertising is getting more important and we want to examine advertisements according **Campaigns** and **Date**.

Popularity of article content

Since we have started with online content, we are interested which factors are crucial for popularity of our article.

We decided to measure the popularity of each article every **hour**, because the reading habits of our readers probably depends on the time of a day. On the other hand, we cannot afford finer granularity because we have to store the data about all the articles.

The article popularity is probably heavily dependent on the quality and topic of the article. This fact we reflect by choosing the articles in finest details according

- **Publication date**
- **Subcategory of article**
- tags - one word description of unlimited domain (no hierarchy)

[1]In Subsection 1.2.3 the bold mark the finest granularity by given dimension

- **Author**

1.3 Conceptual Design of Data Warehouse

1.3.1 Facts

We present following facts according business processes from Subsection 1.2.2.

- Advertisement - poin of view: Added at the end of a day. For each advertisement it would contain sum up information from the whole day.
- Subscriptions - point of view: Represent one subscription from one customer.
- Article popularity - point of view: Every day we store actual information how our articles are popular for every single article.

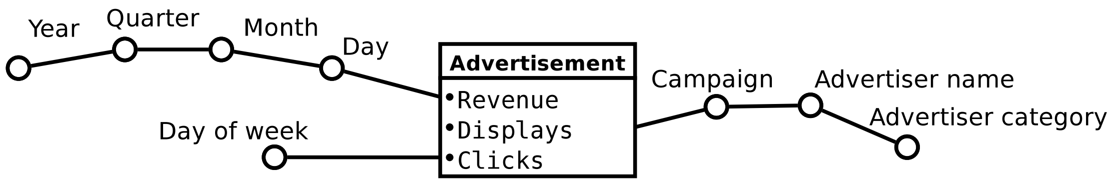


Figure 1.1: Advertisement

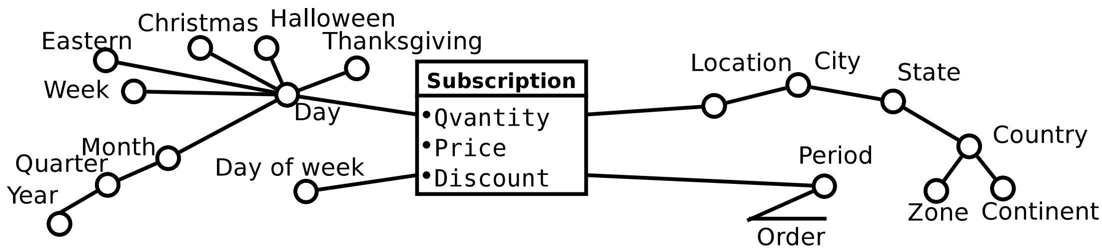


Figure 1.2: Subscriptions

1.3.2 Fact Measures

Advertisement fact has measures *Revenue*, *Displays*, *Clicks*. All 3 measures are additive of an integer type. All three measures could be combined to compute secondary statistic, e.g. Click rate *Clicks/Displays*.

Subscription is measured by *Price*, *Discount* and *Quantity*. The measures simply reflects the act of subscribing per one order. *Quantity* is number of subscriptions in one order. *Price* and *Quantity* are additive measures. *Discount* is proportional e.g. 0.1 meaning 10% discount. It means it is not additive, however an average discount per order makes sense.

Article popularity could be measured by *Number of reads*, *Number of shares* and *Number of comments*. All three measures are additive.

All additive measures in all three facts could be sum up along its hierarchies for additional attributes. The average could be counted using *SUM* and *COUNT* on corresponding level, but realise that average of averages from disjoint subsets is not average from the whole set.

1.3.3 Attributes and Hierarchies

Most of the attributes have self explanatory name. Possibly, the only exception is *Period* dimension in *Subscription* fact. The *Period* describes different types of subscriptions based on how long the subscription last, e.g. month, quarter, annual. The *Period* has attached descriptive attribute *Order*, which it describes the ordering according length, e.g. month has order 1, quarter has order 2, etc.

We managed to get all data which fit our hierarchies completely, so we do not have any optional attributes nor we use any convergences.

Article popularity attributes

The biggest hierarchy is in *Article*, which has attached hierarchies of *Subcategory*, *Category* and the hierarchy *Day*, *Month*, *Year* together with *Author* hierarchy. The *Date* hierarchy in *Article* dimension represents publication date. We use the descriptive attribute *Title* of *Article* to represent one article further on.

We decided to completely separate dimension *Day of week* from *Time* dimension, because we have no interest in rolling up using *Day of week*.

Subscription attributes

In this fact we also separated *Day of a week* and we also used hierarchies *Date*. However, we added *Week* and different *"Holidays"* in the hierarchy after the *Day*. We did not separate it completely, because we are interested which day of month is the most important, so we can aggregate over a day.

Another thing worth to note is that *Order* is a descriptive attribute of *Period*, so we can not aggregate over it.

Advertisement attributes

This is out simplest fact and contains only hierarchy of *Date* and hierarchy of *Campaign*. The *Campaign* hierarchy contains *Advertiser category*, which describes

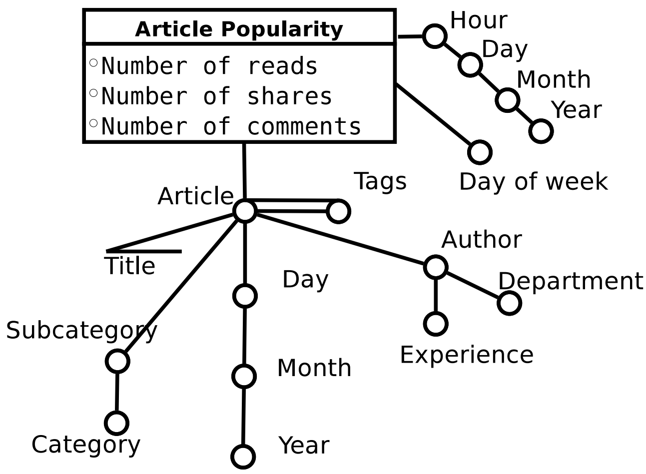


Figure 1.3: Article popularity

how important The Journal managers that the advertiser is. The *Advertiser category* allows us to find out how well we rate our customers in time.

1.3.4 Conceptual problems

At first, we had difficulty identifying facts, but as we discussed business processes and made up sample business queries, we managed to identify the facts.

Secondly, we looked at facts more from a relational point of view. We did not consider evolution of data in time, so we thought about updates in our data warehouse. Especially difficult for us was to avoid updates in facts about articles. We had originally designed the fact that had reflected only actual state. Instead of adding new information and keeping historical data, we had though about updating values in data warehouse. It was clearly wrong solution.

Finally, we managed to figure out how to designed the fact and keep the measures reads, shares and comments and also we know that the tables are sensible large.

1.4 Logical design

1.4.1 Star schemas

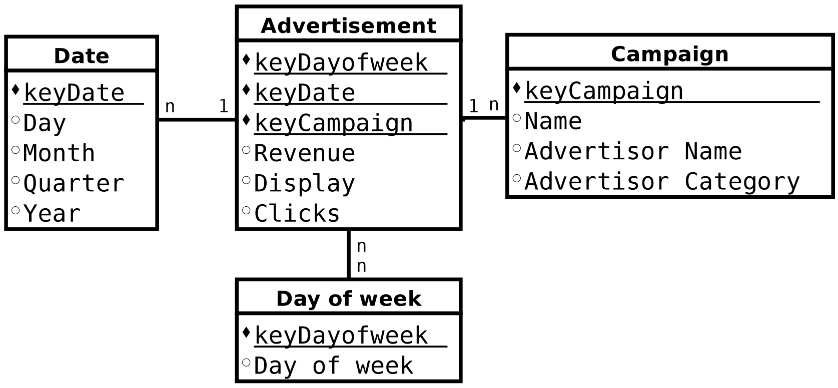


Figure 1.4: Star schema - Advertisement

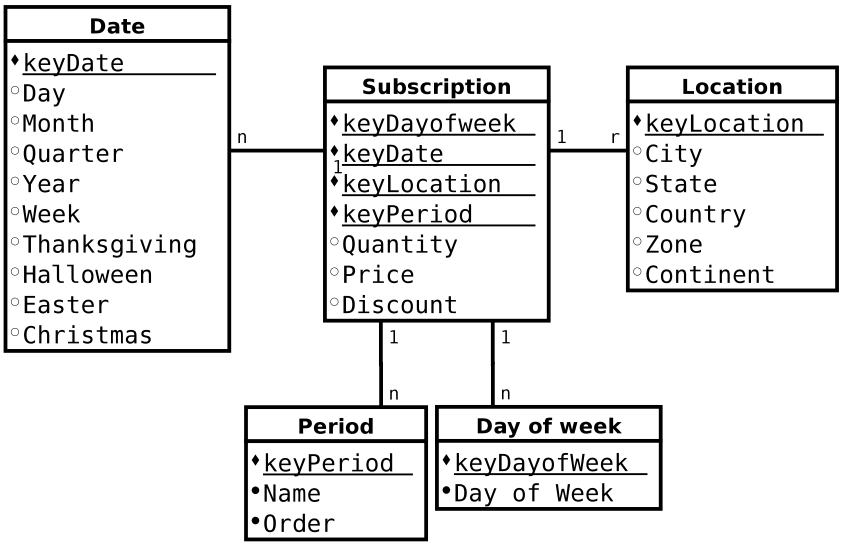


Figure 1.5: Star schema - Subscriptions

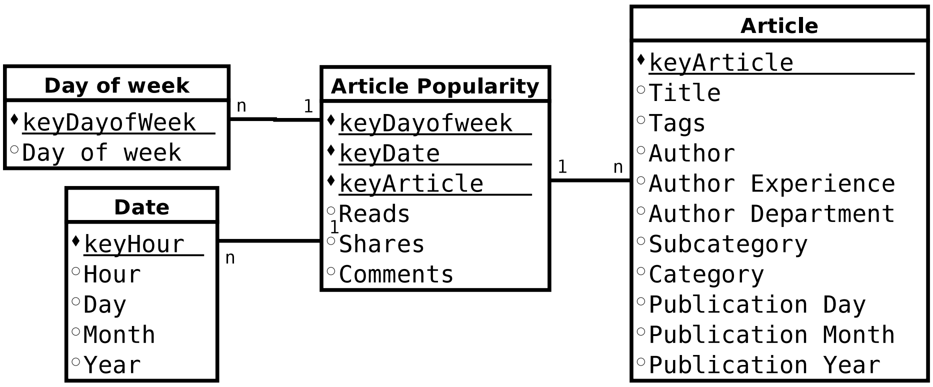


Figure 1.6: Star schema - Article

1.4.2 Snowflake schemas

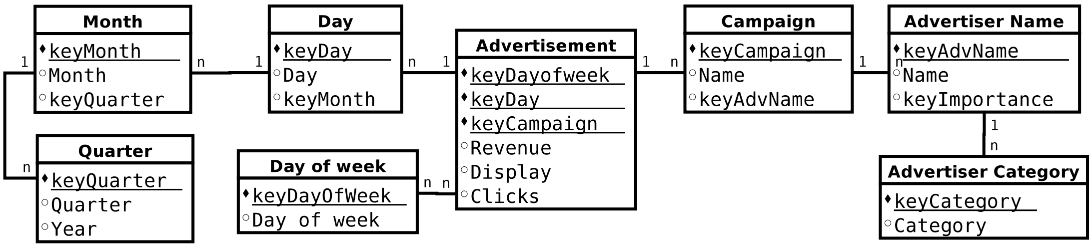


Figure 1.7: Snowflake schema - Advertisement

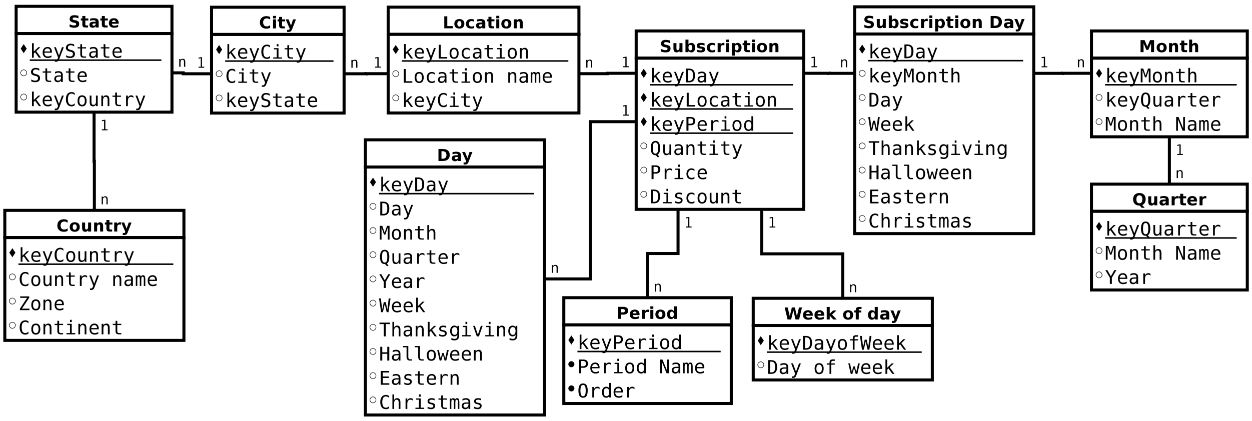


Figure 1.8: Snowflake schema - Subscriptions

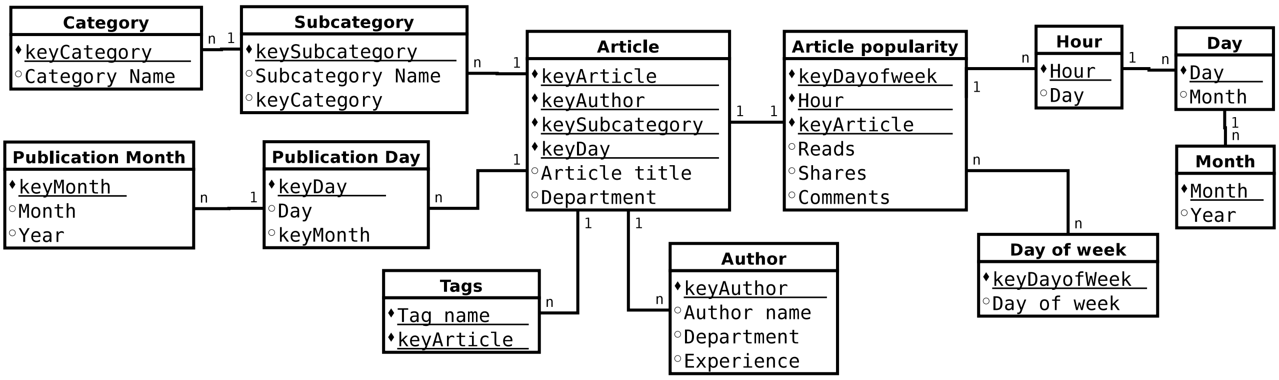


Figure 1.9: Snowflake schema - Article

1.4.3 Choice between schemas

Due to the fact that our dimension tables are not so large, in sense of branching, and also they contain simple domains, the tables in star schema do not cover too much space. The star schema also allows better performance for data warehouse queries. In other words, the low normality of tables does not affect the space requirements too badly, because the numbers of possible values in hierarchies are relatively low.

On the other hand, the star schema allows us process aggregation function more effectively. The aggregation functions together with joins are the key operations for data warehouse. The snowflake schema is in our case split in a lot of tables with relatively small number of values. It does not save so much space compare to star schema, but only it results in sequences of joins. To conclude, we decided to follow star schema and avoid joins in our queries as much as possible.

The only exception, which does not allow us to follow our star schema from Section 1.4.1, is the *Tag* attribute. The *Tag* attribute from *Article popularity* fact has potentially unlimited domain, because every author or editor can add arbitrary tag. We are using only sample data for it, but we still created a separate table for describing n to n relation between Tags and Articles. The table could be very huge and is nonsense to make the table of *Article* dimension wider by another columns as it is displayed in star schema design.

1.4.4 Example data

In tables below you see sample date for a star schema introduced in Subsection 1.4.1. As we mention in Subsection 1.4.3 we did not used in our implementation exactly the star schema, but we separated *tags* attribute to another table. So notice that in our implementation we added a *keyTag* column in Table 1.17, remove column *Tags* in Table 1.19 and we created **separate table** *Tags*.

Figure 1.10: Advertisement fact table

keyDate	keyCampaign	Revenue	Displays	Clicks
874	32	20000	432812	4221
932	872	399	21322	213

Figure 1.11: Advertisement Date dimension table

keyDate	Date	Month	Quarter	Year
874	23	1	1	2008
932	5	5	2	2006

Figure 1.12: Advertisement Campaign dimension table

keyCampaign	Name	Advertiser Name	Advertiser Category
32	2012 Volkswagen CC Ad	Volkswagen	Big Fish
872	Free Lunch at Mensa	Bolzano University	Small Fish

Figure 1.13: Subscription fact table

keyDate	keyLocation	keyPeriod	Quantity	Price	Discount
423	2311	1223	2	100.58	0.1
43	88	8898	10	920.3	0.1

Figure 1.14: Subscription date dimension table

keyDate	Date	Month	Quarter	Year	Week	Thanks-giving	Eastern	Christ-mas	Hallo-ween
423	31	10	4	2011	1	0	0	0	1
43	24	12	4	2005	6	0	0	1	0

Figure 1.15: Subscription location dimension table

keyLocation	City	State	Country	Zone	Continent
2311	New York	New York	United States	US and Canada	North America
88	Bolzano	South Tyrol	Italy	Southern Europe	Europe

Figure 1.16: Subscription period dimension table

keyPeriod	Name	Order
1223	Month	3
8898	Annual	5

Figure 1.17: Article Popularity fact table

keyDate	keyArticle	Reads	Shares	Comments
2011020117	1000	212	20	5
2011020118	1000	53	2	4

Figure 1.18: Article Popularity Date dimension table

keyDate	Hour	Date	Month	Year
2011020117	17	02	11	2011
2011020118	18	02	11	2011

Figure 1.19: Article Popularity dimension table

key Article	Title	Tags	Author	Author Exp.	Author Dept.	Sub category	Category	Pub. Day	Pub. Month	Pub. Year
1000	Article	rude funny	John Smith	Junior Reporter	Entertainment	Fashion	Life and Style	02	11	2011
9999	Wake the economy	serious actual	Jane Doe	Senior Reporter	Finance	Earnings	Business	03	11	2011

1.5 SQL Implementation

Our data was generated partially randomly and partially randomly chosen from public databases. Firstly, we created the populating scripts for PostgresSQL 9.1. Lately, we decided to migrate our database to Oracle 10.2.0.3.0, because Postgres does not support data mining extensions. We had to modify the populating scripts to fit Oracle syntax and also subdivide them, because Oracle does not support some commands with our permissions, e.g. "CREATE SCHEMA" command.

We divided generated data into several SQL scripts which creates tables, populates tables, and delete tables. We provide init.sh file. It allows to populate database, see Figure 1.20, and also deleting the tables, see Figure 1.21.

```
$ cd init
# comment: init connects to database granted to us from school
# comment: for more information check the script itself
$ ./init.sh init login password
# it will take quite about 4 hours to populate the data
```

Figure 1.20: Populating data warehouse using init.sh script

```
$ cd init
# comment: droping the table/ removing the data warehouse
# comment: still from init directory
$ ./init.sh drop login password
```

Figure 1.21: Removing data warehouse using init.sh script

For more details about SQL scripts, the init.sh script or the script for generating data view the source code of the scripts, it should be self explenatory. In enclosed README.txt we list all containing scripts and the usage again.

Data

To generate the SQL scripts for populating the database we used PHP scripts enclosed.

We created some data artificially, for some we used real world data. It is worth to note that we used real address data from Canada and also we filled the tags with a few samples from real tags from real Wall Street Journal.

2. Milestone - Data warehouse querying

2.1 Content of Milestone 2

In milestone 2 we picked up 20 business queries which correlates to main process described in Section 1.2.2. You can find the queries in written and SQL form in Section 2.2. We have separately described special queries in Subsections 2.2.1, 2.2and in 2.2.2.

In second part of this milestone we will improve performance of 3 queries from Section 2.2 by using materialized views and indexes.

We have decided to improve performance of 2 queries at first by using materialized vies in Subsection 2.3.1 and later using indexes in Subsection 2.3.2. In both sections we have chosen different third query. At the end of the Section 2.3 we compare the performance of first 2 queries using indexes and materialized views.

2.2 Business queries

Subscription queries

If the row number is not explicitly specified than full results are presented.

1. Revenue from subscriptions by year?

```
select sum(price * (1-discount) * quantity) revenue, d.year
from sub_subscription s
join sub_date d on s.keydate = d.keydate
group by d.year
order by year desc;
```

2.444s

	REVENUE	YEAR
1	58063611.94955	2011
2	60889481.4008	2010
3	2537349.7857	2009

2. Which were the most popular subscription periods each year?

```
select year, p.name as period, sum(quantity) subscriptions
from oplatek.sub_subscription s
join oplatek.sub_date d on s.keydate = d.keydate
join oplatek.sub_period p on s.keyperiod = p.keyperiod
group by d.year, p.name
order by year desc, subscriptions desc;
```

0.808s

	YEAR	PERIOD	SUBSCRIPTIONS
1	2011	Year	274708
2	2011	Week	274274
3	2011	Day	273202
4	2011	Quarter	272403
5	2011	Month	271001
6	2010	Quarter	287861
7	2010	Year	287427
8	2010	Week	285587
9	2010	Month	283808
10	2010	Day	283459

3. Revenue from subscriptions and subscription count by country in 2010?

```
select l.country, sum(price * (1-discount) * quantity) revenue,
  sum(quantity) subscriptions
from oplatek.sub_subscription s
join oplatek.sub_location l on s.keylocation = l.keylocation
join oplatek.sub_date d on s.keydate=d.keydate and d.year=2010
group by l.country
order by revenue desc;
```

0.639s

	COUNTRY	REVENUE	SUBSCRIPTIONS
1	Italy	4306089.4312	101055
2	Ireland	4154497.9457	96061
3	Sweden	4143242.65915	97048
4	United Kingdom	4024441.86145	94005
5	China	3902442.5813	91839
6	Norway	3888759.8255	91400
7	United States	3732016.12195	87939
8	Canada	3447476.77135	82527
9	Spain	3248450.4486	76637
10	Poland	3059984.2539	71159

4. Top 10 cities from each country with the highest revenue from subscriptions in 2010?

```
select * from (select country, city,
  sum(price * (1-discount) * quantity) revenue,
  rank() over(partition by country
    order by sum(price * (1-discount) * quantity) desc) rank
from oplatek.sub_subscription s
join oplatek.sub_date d
  on s.keydate = d.keydate and d.year = 2010
join oplatek.sub_location l
  on s.keylocation = l.keylocation
group by l.country, l.city
order by revenue desc)
where rank <= 10;
```

50 rows 0.711s

	COUNTRY	CITY	REVENUE	RANK
1	Germany	City #231	496463.60295	1
2	Italy	City #73	489745.0696	1
3	Ireland	City #99	479657.3802	1
4	United States	City #82	478444.16645	1
5	United Kingdom	City #63	474282.5723	1
6	China	City #147	467170.10765	1
7	United Kingdom	City #54	467074.7586	2
8	United Kingdom	City #189	466046.36855	3
9	Germany	City #148	463341.58105	2
10	Canada	City #25	460309.5905	1

5. How much would we earn without applying discounts on subscriptions, by period type and by year?

```
select year, period, revenue, revenue_no_discounts,
  (revenue_no_discounts - revenue) difference from
    (select d.year, p.name as period,
     sum(price * (1-discount) * quantity) revenue,
     sum(price * quantity) revenue_no_discounts
    from oplatek.sub_subscription s
    join oplatek.sub_date d on s.keydate = d.keydate
    join oplatek.sub_period p on s.keyperiod = p.keyperiod
    group by d.year, p.name)
order by year desc, period asc;
```

1.127s

	YEAR	PERIOD	REVENUE	REVENUE_NO_DISCOUNTS	DIFFERENCE
1	2011	Day	543671.98	543671.98	0
2	2011	Month	4311727.485	4333305.99	21578.505
3	2011	Quarter	13501390.1818	13617425.97	116035.7882
4	2011	Week	1638809.64075	1642901.26	4091.61925
5	2011	Year	38068012.662	38456372.92	388360.258
6	2010	Day	564083.41	564083.41	0
7	2010	Month	4515723.108	4538089.92	22366.812
8	2010	Quarter	14268186.7918	14390171.39	121984.5982
9	2010	Week	1706521.649	1710666.13	4144.481
10	2010	Year	39834966.442	40236905.73	401939.288

7. Revenue by month and by state in Canada in 2010 together with average revenue by states in Canada in the same month?

```
select l.state, d.month,
  sum(price * (1-discount) * quantity) revenue,
  /* avg_revenue_in_this_month_across_all_states */
  avg(sum(price * (1-discount) * quantity))
    over (partition by month) avg_rev
from oplatek.sub_subscription s
join oplatek.sub_location l
    on s.keylocation = l.keylocation and l.country = 'Canada'
join oplatek.sub_date d
    on s.keydate = d.keydate and d.year = 2010
where length(l.state) = 2
group by l.state, d.month
order by state asc, month asc;
```

156 rows 0.711s

	STATE	MONTH	REVENUE	AVG_REV
1	DA	1	5967.5335	24449.9297923076923076923076923077
2	DA	2	2145.6436	19624.4675153846153846153846153846
3	DA	3	1498.6315	23564.5622807692307692307692307692
4	DA	4	2224.4116	21621.8326153846153846153846153846
5	DA	5	4015.37345	21000.9800615384615384615384615385
6	DA	6	2556.5805	20106.9801038461538461538461538462
7	DA	7	2634.36135	22503.9142615384615384615384615385
8	DA	8	4761.28285	22445.4762769230769230769230769231
9	DA	9	2873.75625	21041.4624615384615384615384615385
10	DA	10	2835.05085	23492.89075
11	DA	11	2101.2405	22700.3226038461538461538461538462
12	DA	12	2326.941	22637.70215
13	FO	1	23329.1536	24449.9297923076923076923076923077
14	FO	2	20776.74385	19624.4675153846153846153846153846
15	FO	3	26776.31285	23564.5622807692307692307692307692
16	FO	4	18953.7561	21621.8326153846153846153846153846
17	FO	5	26305.70565	21000.9800615384615384615384615385
18	FO	6	15416.2112	20106.9801038461538461538461538462
19	FO	7	27026.62355	22503.9142615384615384615384615385

Article queries

1. Top 10 read articles and their authors for every month in year 2011?

```
select * from (
 select d.month, a.keyarticle as id,
  a.title, a.author, sum(reads) reads,
  rank() over (partition by month order by sum(reads) desc) rank
 from oplatek.artpop_articlepopularity f
 join oplatek.artpop_article a on f.keyarticle = a.keyarticle
 join oplatek.artpop_date d on f.keydate = d.keydate
     and d.year = 2010
 group by d.month, a.keyarticle, a.title, a.author
 order by month asc, reads desc
) where rank <= 10;
```

121 rows 2.604s

	MONTH	ID	TITLE	AUTHOR	READS	RANK
1	1	4	Article #4	Author #80	1644	1
2	1	11	Article #11	Author #55	1641	2
3	1	5	Article #5	Author #68	1608	3
4	1	3	Article #3	Author #17	1605	4
5	1	7	Article #7	Author #163	1600	5
6	1	18	Article #18	Author #42	1593	6
7	1	9	Article #9	Author #23	1582	7
8	1	8	Article #8	Author #195	1580	8
9	1	13	Article #13	Author #78	1580	8
10	1	6	Article #6	Author #154	1575	10

2. Hours of the day when most articles are read grouped by category in year 2010 together with the average number of articles read during this hour in the same year?

```
select a.category, d.hour, sum(reads) reads,
    avg(sum(reads)) over (partition by hour) avg_reads
from oplatek.artpop_articlepopularity f
```

```
join oplatek.artpop_article a on f.keyarticle = a.keyarticle
join oplatek.artpop_date d on f.keydate = d.keydate
    and d.year = 2010
group by a.category, d.hour
order by a.category asc, d.hour asc;
```

96 rows 0.415s

	CATEGORY	HOUR	READS	AVG_READS
1	Business	0	3016	3277.75
2	Business	1	3143	3289.25
3	Business	2	3089	3261
4	Business	3	3061	3320.5
5	Business	4	2972	3226.25

3. Number of articles published in each subcategory in year 2010 together with the percentage of total articles in the category?

```
select category, subcategory,
 articles/sum(articles) over (partition by category) perct,
 articles,
 sum(articles) over (partition by category) as articles_in_category
 from (
  select a.category, a.subcategory, count(a.keyarticle) articles
  from oplatek.artpop_articlepopularity f
  join oplatek.artpop_article a
   on f.keyarticle = a.keyarticle and a.publicationyear = 2011
  group by a.category, a.subcategory
  order by category, subcategory
 );
```

1.23s

	CATEGORY	HOUR	READS	AVG_READS
1	Business	0	3016	3277.75
2	Business	1	3143	3289.25
3	Business	2	3089	3261
4	Business	3	3061	3320.5
5	Business	4	2972	3226.25
6	Business	5	3059	3225.75
7	Business	6	3021	3230
8	Business	7	3059	3247.5

4. Top 5 authors in every category by comments on their articles?

```
select * from (
select a.category, a.author, sum(comments) comments,
  rank() over (partition by a.category
    order by sum(comments) desc) rank
from oplatek.artpop_articlepopularity f
join oplatek.artpop_article a on f.keyarticle = a.keyarticle
group by a.category, a.author
order by a.category asc, comments desc
) where rank <= 5;
```

0.823 s

	CATEGORY	AUTHOR	COMMENTS	RANK
1	Business	Author #121	1654	1
2	Business	Author #124	1577	2
3	Business	Author #36	1392	3
4	Business	Author #116	1331	4
5	Business	Author #75	1327	5
6	Life & Style	Author #142	1872	1
7	Life & Style	Author #37	1712	2
8	Life & Style	Author #17	1672	3
9	Life & Style	Author #179	1483	4
10	Life & Style	Author #154	1404	5
11	Tech	Author #64	1353	1
12	Tech	Author #193	1249	2
13	Tech	Author #68	1245	3
14	Tech	Author #145	1206	4
15	Tech	Author #131	1200	5
16	World	Author #118	1523	1
17	World	Author #30	1423	2
18	World	Author #18	1378	3
19	World	Author #99	1372	4

5. Top 20 articles published in October, 2011 with the #comments/#reads higher than the average #comments/#reads having at least 20000 #reads?

```
select * from (
select a.keyarticle id, a.title,
    sum(reads)/sum(comments) comm_by_reads,
    avg(sum(reads)/sum(comments)) over () avg_comm_by_reads,
    sum(reads) reads
from oplatek.artpop_articlepopularity f
join oplatek.artpop_article a on f.keyarticle = a.keyarticle
where a.publicationyear = 2011
and a.publicationmonth = 10
group by a.keyarticle, a.title
order by comm_by_reads - avg_comm_by_reads desc
) where comm_by_reads > avg_comm_by_reads
and reads >= 20000 and rownum <= 20;
```

1.125 s

	ID	TITLE	COMM_BY_READS	AVG_COMM_BY_READS	READS
1	96	Article #96	732.691489361702127659574468085106382979	416.45282416681304060684332010071659871	68873
2	10	Article #10	687.723880597014925373134328358208955224	416.45282416681304060684332010071659871	184310
3	15	Article #15	676.077205882352941176470588235294117647	416.45282416681304060684332010071659871	183893
4	17	Article #17	656.913669064748201438848920863309352518	416.45282416681304060684332010071659871	182622
5	8	Article #8	656.236559139784946236559139784946236559	416.45282416681304060684332010071659871	183090
6	11	Article #11	635.8819444444444444444444444444444444444	416.45282416681304060684332010071659871	183134
7	678	Article #678	634.692307692307692307692307692307692308	416.45282416681304060684332010071659871	66008
8	16	Article #16	631.871972318339100346020761245674740484	416.45282416681304060684332010071659871	182611
9	5	Article #5	627.257525083612040133779264214046822742	416.45282416681304060684332010071659871	187550
10	67	Article #67	618.136363636363636363636363636363636364	416.45282416681304060684332010071659871	67995
11	12	Article #12	610.667774086378737541528239202657807309	416.45282416681304060684332010071659871	183811
12	389	Article #389	608.8125	416.45282416681304060684332010071659871	68187
13	6	Article #6	608.775919732441471571906354515050167224	416.45282416681304060684332010071659871	182024
14	425	Article #425	607.891891891891891891891891891891891892	416.45282416681304060684332010071659871	67476
15	7	Article #7	600.139158576051779935275080906148867314	416.45282416681304060684332010071659871	185443
16	1498	Article #1498	595.425925925925925925925925925925925926	416.45282416681304060684332010071659871	64306
17	66	Article #66	583.358333333333333333333333333333333333	416.45282416681304060684332010071659871	70003
18	548	Article #548	581.556521739130434782608695652173913043	416.45282416681304060684332010071659871	66879
19	13	Article #13	580.382716049382716049382716049382716049	416.45282416681304060684332010071659871	188044

6. Compare the number of reads/shares/comments of articles tagged with tags 'positive' and 'negative' for each year?

```
select year, tag, reads,
 sum(reads) over (partition by year) reads_total,
 reads/sum(reads) over (partition by year) reads_perc,
 reads, sum(shares) over (partition by year) shares_total,
 shares/sum(shares) over (partition by year) shares_perc,
 reads, sum(comments) over (partition by year) comments_total,
 comments/sum(comments) over (partition by year) comments_perc
from (
 select a.publicationyear year, t.tag, sum(reads) reads,
  sum(comments) comments, sum(shares) shares
 from oplatek.artpop_articlepopularity f
 join oplatek.artpop_article a on f.keyarticle = a.keyarticle
 join oplatek.artpop_tag t on t.tag in ('positive', 'negative')
 join oplatek.artpop_articletags artt
  on a.keyarticle = artt.keyarticle and artt.keytag = t.keytag
 group by a.publicationyear, t.tag
 order by year desc, tag
);
```

0.611 s

	YEAR	TAG	READS	READS_TOTAL	READS_PERC	READS_1	SHARES_TOTAL	SHARES_PERC
1	2011	positive	51342051	101057724	0.5080467773052161752623678720490479282 91	51342051	502601	0.5073169372922
2	2011	negative	49715673	101057724	0.4919532226947838247376321279509520717 09	49715673	502601	0.4926830627077

7. Top 100 tags by article count in Business category in 2011?

```
select rank() over (order by count(a.keyarticle) desc) rank,
 t.tag, count(a.keyarticle) articles
from oplatek.artpop_articlepopularity f
join oplatek.artpop_article a
 on f.keyarticle = a.keyarticle
 and a.category = 'Business' and a.publicationyear = 2011
join oplatek.artpop_tag t on 1=1
join oplatek.artpop_articletags artt
 on a.keyarticle = artt.keyarticle and artt.keytag = t.keytag
group by t.tag
order by rank, tag asc;
```

0.797 s

	RANK	TAG	ARTICLES
1	1	short	170094
2	2	neutral	154677
3	3	long	111576
4	4	positive	110979
5	5	medium	110254
6	6	negative	104164

Advertisement queries

1. Revenue by year together with average revenue in all years together with revenue in this year and together with last year?

```
select ad.Year, sum(aa.revenue),
  avg(sum(aa.revenue)) over () as total_avg,
  sum(sum(aa.revenue)) over
    (order by ad.year rows 1 preceding) as sum_last_2_years
from
advert_advertisement aa
join advert_date ad on aa.keydate = ad.keydate
group by ad.YEAR;
```

0.155 s

	YEAR	SUM(AA.REVENUE)	TOTAL_AVG	SUM_LAST_2_YEARS
1	2006	38637.69	236213.505	38637.69
2	2007	291053.43	236213.505	329691.12
3	2008	289185.45	236213.505	580238.88
4	2009	293761.71	236213.505	582947.16
5	2010	294360.24	236213.505	588121.95
6	2011	210282.51	236213.505	504642.75

2. CPM(Clicks divided by displays) for top 10 advertisers by revenue together with avg CPM for advertisers category having the advertiser at least 15 campaigns?

```
select *
from
( select
      rank() over (order by sum (aa.revenue) desc) as top,
      sum(aa.clicks)/sum(aa.displays),ac.advertiserName,
      ac.advertiserCategory as cat,
      avg(sum(aa.clicks)/sum(aa.displays))
       over (partition by ac.advertiserCategory)
  from advert_advertisement aa
  join advert_campaign ac on aa.keycampaign = ac.keycampaign
  join
  ( select distinct in_aa.advertiserName
    from advert_campaign in_aa
    group by in_aa.advertiserName
    having COUNT(in_aa.name) >= 15
  )
  camp on ac.advertiserName = camp.advertiserName
  group by rollup (ac.advertiserName,ac.advertiserCategory)
  having grouping_id(ac.advertiserName,ac.advertiserCategory)=0
  )
where
top < 10;
```

0.643 s

	TOP	SUM(AA.CLICKS)/SUM(AA.DISPLAYS)	ADVERTISERNAME	CAT	AVG(SUM(AA.CLICKS)/SUM(AA.DISPLAYS))OVER(PARTITIONBYAC.
1	1	0.005058192525575667061765385417964430315347	Advertiser #259	Top Client	0.0049835317509954688937
2	2	0.005043346520304733309008697557018968474827	Advertiser #359	Top Client	0.0049835317509954688937
3	3	0.004914495021129408068296706385455339531268	Advertiser #499	Top Client	0.0049835317509954688937
4	4	0.00498566408586392893903167250842890002618	Advertiser #301	Top Client	0.0049835317509954688937
5	5	0.004901205585957958890211815741751362067511	Advertiser #199	Top Client	0.0049835317509954688937
6	6	0.00492250532878984424703075903148524207254	Advertiser #256	Top Client	0.0049835317509954688937
7	7	0.005014626224176025795180440503010050976934	Advertiser #415	Top Client	0.0049835317509954688937
8	8	0.004972130677080678176689002501250574838129	Advertiser #209	Top Client	0.0049835317509954688937
9	9	0.004998935484690788612318945526694629998371	Advertiser #344	Top Client	0.0049835317509954688937

3. Revenue by advertiser from "Small Fish" category who has greater revenue than average of the "Middle Fish" bias=0.5 together with average of "Middle Fish" advertisers?

```
select ac.advertiserName, ac.advertiserCategory, sum(aa.revenue),
 avg(sum(aa.revenue)) over (partition by ac.advertiserCategory)
 as small_fish_avg,
 (
  select distinct
   AVG(sum(aaa.revenue)) over
    (partition by aac.advertiserCategory) as middle_fish_avg
    from advert_advertisement aaa
    join advert_campaign aac on aaa.keycampaign=aac.keycampaign
    where aac.advertiserCategory = 'Medium Fish'
    group by aac.advertiserName,aac.advertiserCategory
  ) as middle_fish_avg
from advert_advertisement aa
join advert_campaign ac on aa.keycampaign = ac.keycampaign
where ac.advertiserCategory = 'Small Fish'
group by ac.advertiserName,ac.advertiserCategory
having sum(aa.revenue) > 0.5* (
  -- having is stupid, I have to repeat query:)
  select distinct
    AVG(sum(aaa.revenue))
      over (partition by aac.advertiserCategory)
    as middle_fish_avg
  from advert_advertisement aaa
  join advert_campaign aac on aaa.keycampaign = aac.keycampaign
  where aac.advertiserCategory = 'Medium Fish'
  group by aac.advertiserName,aac.advertiserCategory);
```

0.301s

	ADVERTISERNAME	ADVERTISERCATEGORY	SUM(AA.REVENUE)	SMALL_FISH_AVG	MIDDLE_FISH_AVG
1	Advertiser #112	Small Fish	1288.53	1268.42	2368.92675
2	Advertiser #487	Small Fish	1263.93	1268.42	2368.92675
3	Advertiser #78	Small Fish	1252.8	1268.42	2368.92675

4. The Campaigns which lasted more than 5 month with revenue bigger than 140 at least in one from the 5 month, all in year 2011?

```
select * from (
  select ac.name, max(sum(aa.revenue))
```

```
    over (partition by ac.name) as max_revenue_over_month
  from advert_advertisement aa
  join advert_date ad on ad.keyDate = aa.keyDate
  join advert_campaign ac on ac.keyCampaign = aa.keyCampaign
  where ad.year=2011
  group by ac.name, ad.month
  having ( max(ad.month) - min(ad.month)) >= 0
    or ( (max(ad.month) - min(ad.month)) = 5
    and (max(ad.day) - min(ad.day)) >=0 )
) where max_revenue_over_month > 140;
```

0.16 s

	NAME	MAX_REVENUE_OVER_MONTH
1	Campaign #25 for Advertiser #13	144.09
2	Campaign #25 for Advertiser #13	144.09
3	Campaign #44 for Advertiser #237	145.68
4	Campaign #44 for Advertiser #237	145.68
5	Campaign #72 for Advertiser #461	153.99

5. Most popular campaigns by year together with their popularity(clicks+displays) and avg CPM per month?

```
select y, c, n, popularity, cpm_per_month
from (
  select ad.year y, ac.name n, ad.month, ac.advertisercategory c
  , sum(aa.clicks+aa.displays) popularity
  ,(avg(sum(aa.clicks)/sum(aa.displays))
    over (partition by ad.month order by ad.month
    rows between 1 preceding and 1 following)) cpm_per_month
  from advert_advertisement aa
  join advert_date ad on ad.keyDate = aa.keyDate
  join advert_campaign ac on ac.keyCampaign = aa.keyCampaign
  group by ad.month,ad.year, ac.name,ac.advertisercategory
  )
order by y desc, c, popularity desc, n;
```

0.61 s, 35889 rows

	Y	C	N	POPULARITY	CPM_PER_MONTH
1	2011	Big Fish	Campaign #5 for Advertiser #222	860887	0.004661016036467124135044726986752405801767
2	2011	Big Fish	Campaign #72 for Advertiser #461	860516	0.005832496172358619302072484903213402239833
3	2011	Big Fish	Campaign #25 for Advertiser #13	832966	0.005245587260004505288114497275409173897567
4	2011	Big Fish	Campaign #25 for Advertiser #254	807939	0.0046432452003766594237693276129197395131
5	2011	Big Fish	Campaign #82 for Advertiser #341	807295	0.0053392457693738247654763792390342070645
6	2011	Big Fish	Campaign #44 for Advertiser #237	806360	0.005829351528764098739987205450966629677933
7	2011	Big Fish	Campaign #29 for Advertiser #101	795873	0.0042929765322644977623177950636692411126
8	2011	Big Fish	Campaign #86 for Advertiser #358	794679	0.0046730897934679767061759949409578962085
9	2011	Big Fish	Campaign #18 for Advertiser #242	792328	0.003067924265933017947019262593757108215575
10	2011	Big Fish	Campaign #52 for Advertiser #125	790447	0.0059227748684711894439508138795212606648
11	2011	Big Fish	Campaign #71 for Advertiser #210	788699	0.005194144003355359737447665985682044865633
12	2011	Big Fish	Campaign #27 for Advertiser #458	786455	0.005510808154377612778377084152621200072133
13	2011	Big Fish	Campaign #53 for Advertiser #69	780134	0.005109568677884844377230393721913898598833
14	2011	Big Fish	Campaign #5 for Advertiser #42	763556	0.0052406808930762681770158961439525511654
15	2011	Big Fish	Campaign #42 for Advertiser #201	762836	0.0043892980547799572930469235437493292407
16	2011	Big Fish	Campaign #47 for Advertiser #331	761343	0.003748270348066450065113237689684919342033
17	2011	Big Fish	Campaign #61 for Advertiser #339	760819	0.0057850089240451726538696913930255649786
18	2011	Big Fish	Campaign #9 for Advertiser #160	760527	0.004518377475911881113642081598537871420067
19	2011	Big Fish	Campaign #18 for Advertiser #92	760315	0.004870185025038734696535504250487659215133

Ranking queries

We have enclosed our ranking queries in list of business queries at Subsection 2.2. See queries Subscription query number 4, from Article queries numbers 2, 4, 5 and query 2 from Advertisement section

2.2.1 Windowing queries

We used windowing queries in 1 and 5 queries in Advertisement part from Subsection 2.2.

2.2.2 Period-to-Period queries

- Revenue from subscriptions by year and week compared to the revenue on the same week in the previous year?

```
select year, week, revenue,
 sum(revenue) over (partition by week order by year
  RANGE BETWEEN 1 PRECEDING AND 1 PRECEDING)
   revenue_last_year_this_week
from (
 select
  year, week, sum(price * (1-discount) * quantity) revenue
 from oplatek.sub_subscription s
 join oplatek.sub_date d on s.keydate = d.keydate
 group by d.year, d.week)
order by year desc, week asc;
```

- Revenue from subscriptions by year and week compared to the average revenue on the same week in the 3 previous years?

```
select year, week, revenue, avg(revenue)
 over (partition by week order by year
        RANGE BETWEEN 3 PRECEDING AND 1 PRECEDING)
         avg_revenue_prev_3_years
```

```
from (
select year, week,
 sum(price * (1-discount) * quantity) revenue
from oplatek.sub_subscription s
join oplatek.sub_date d on s.keydate = d.keydate
group by d.year, d.week)
order by year desc, week asc;
```

2.2.3 Dense reports

- Revenue from subscriptions by year?

```
select b.year, NVL(revenue, 0) dense_revenue from (
select year, sum(price * (1-discount) * quantity) revenue
from oplatek.sub_subscription s
join oplatek.sub_date d on s.keydate = d.keydate
group by d.year
order by year desc
) a
right outer join (
  select distinct dd.year
  from oplatek.sub_date dd
) b on (a.year = b.year)
order by b.year desc;
```

- Number of articles published in 1999 in each category?

```
select b.category, b.subcategory, NVL(articles,0) dense_articles
from (
 select a.category, a.subcategory, count(a.keyarticle) articles
 from oplatek.artpop_articlepopularity f
 join oplatek.artpop_article a
  on f.keyarticle = a.keyarticle and a.publicationyear = 1999
 group by a.category, a.subcategory
 order by category
) a
right outer join (
  select distinct aa.category, aa.subcategory
  from oplatek.artpop_article aa
) b
on (a.category = b.category and a.subcategory = b.subcategory)
order by category, subcategory;
```

2.3 Execution plan

We have chosen fact Subscription for implementing materialized views and indexes. We will implement materialised views for queries number 3 4 and 7 from Subscription part of Section 2.2.

The three above mentioned queries are good candidates for materialized view because they aggregate heavily and gives relative small results based on lot of data.

Also for indexes is this fact interesting because the queries 3 and 4 use "where" clause on columns with no indexes so implementing indexes will result in better performance.

Last but definitely not least, we suppose that Subscription fact would be the most queried fact from our data warehouse.

It is worth to mention that we decided to choose third query for impementing indexes for fact article popularity to see the difference. The main reason is the fact has 1672252 rows, so should be able to see the performance improvement easily.

2.3.1 Materialized views

In our queries we do not use dimensions *Period Name* and *Day of week*, but we use *Date* hierarchy and *Location* hierarchy. In order to discuss, which materialised views should we implement we will use Lattice framework to describe dependencies. We will denote the size of views directly in the lattice diagram for relevant views.

The "*group by*" sets needed to answer all 3 queries are represented by all nodes except the node representing the finest granularity. So the candidate views cover the whole lattice except the top most node on Figure 2.8.

Figure 2.1: Attributes in group by statement from our 3 queries

Query 3	Query 4	Query 7
Country	City, State	State, Month

All views which has at least the same level of aggregation as our chosen 3 queries in one attribute are relevant candidates for materialised views. Unfortunately, aggregation of our 3 queries (See Figure 2.1) covers almost whole lattice.[1]. So we have to decide among large number of candidates.

We have created 3 materialised views 2.2, 2.4 and 2.6. The views are denoted in Figure 2.8. See the lattice legend to find out that *Co* means that the view is grouped by a country. Queries in Figures 2.3, 2.5 and 2.7 use the views and correspond to queries 3, 4 and 7.

(We have decided to store in our views all ready precomputed aggregation values, because the computed measure *revenue* is additive. It arise a possibility to use a View from Figure 2.4 in rewritten queries 2.3 and 2.5, because we aggregate along location hierarchy. The table below describes performance improvements for separate views.

[1]The lattice represents the level of aggregation

View	Rows in View	Size[MB]	Query 3[s]	Query 4[s]	Query 7[s]
No view	0	-	0.018	0.031	0.109
$\{S, M\}$	$\|Month\| * \|State\|$	0.0625	-	-	0.056
$\{Ci\}$	$\|City\|$	0.0625	!0.9!-	0.18	-
$\{Co\}$	$\|Country\|$	0.0625	0.01	-	-

The value for view $\{Ci\}$ and query 4 is marked with exclamation marks, because it does not return complete results. We included it to explore how the performance change using different level of aggregation in views. In the summary at the end of this subsection we stress, that we can afford not to reuse the vies, because there are ridiculously small.

During writing the queries using views, we realised that star schema does not allow easily combine "grouped-by" tables from different dimensions easily. In fact, to use two separate views with aggregated values by *month* and in second view aggregated values by *state*, we would be forced to join the dimension tables of *location* and *date* containing the whole hierarchies.

Such table is bigger than join of the $facttable$ with the *date* and *location* dimension, so it is nonsense to use it. If we had implemented snowflake schema, the we can join more tables along hierarchies but only with the key values, so it would make sense to measure the performance in that case.

Summary

To conclude, this section about choosing and implementing materialised views we would like to stress some observation which we have learned from our data.

- For queries which **aggregates heavily** and which has results a few rows, it **always** pays of to use materialised views. In our example it is views $\{S, M\}$ and $\{Co\}$ see table above.
- All our Views including does not exceeds the smallest default block allocated by Oracle [2]
- Views speed up queries a lot, but if you do reuse the views in different queries, caching is probably easier alternative and works really well.

[2] We used table *user_extents* to determine size of tables. See section 1) in Figure 2.9.

```
create materialized view view_co
build immediate as
select l.country,
  sum( s.price * (1-s.discount) * s.quantity ) s_revenue,
  sum(quantity) s_subscriptions
from sub_subscription s
join sub_date d on s.keyDate = d.keyDate
join sub_location l
  on s.keyLocation = l.keyLocation and d.year=2010
group by l.country;
```

Figure 2.2: Materialized view $\{Co\}$ grouped by *country*

```
-------- rewriten for view_co ->0.02s
select s_revenue , s_subscriptions from view_co
order by s_revenue desc;
-------- rewritten for view_ci
--- BAD RESULTS ->PERFORMANCE TEST ONLY
select l.country , sum(ci_revenue) revenue from
view_ci
join sub_location l on l.city = view_ci.city
group by country
order by revenue;
```

Figure 2.3: Rewritten queries for 3. query

```
create materialized view view_ci
build immediate as
select l.city , sum( s.price * (1-s.discount)
* s.quantity ) ci_revenue
from sub_subscription s
join sub_date d on s.keyDate = d.keyDate
join sub_location l on s.keyLocation = l.keyLocation and d.year=2010
group by l.city;
```

Figure 2.4: Materialized view $\{Ci\}$ grouped by *city*

```
--- rewritten for view_ci --0.018
select * from (
 select l.country , l.city , max(ci_revenue),
  rank() over(partition by country
    order by sum(ci_revenue) desc) rank1
 from view_ci
 join sub_location l on view_ci.city = l.city
 group by l.country , l.city
 order by max(ci_revenue) desc)
where rank1 <=10;
```

Figure 2.5: Rewritten query for 4. query

- Due to caching we have to "rename" our queries, because we did not have sufficient privileges to turn caching off[3]
- During design phase possibility of integrating views in star or snowflake schema should play role in deciding between snowflake and star schema.
- Oracle is capable of finding out itself that we used materialized views. (We have to specify[4] it during creation.) On the other hand, it is not working very well, so basically you have to rewrite your queries.

[3] See commands in section 2) in Figure 2.9.

[4] See section 3) in Figure 2.9.

```
create materialized view view_m_s
build immediate as
select l.state, d.month,
  sum( s.price * (1-s.discount)  * s.quantity ) m_s_revenue,
  avg(sum( price * (1-discount) * quantity))
    over (partition by month) m_s_avg_rev
  from sub_subscription s
join sub_date d on s.keyDate = d.keyDate
join sub_location l on s.keyLocation = l.keyLocation
where d.year=2010
group by l.state, d.month;
```

Figure 2.6: Materialized view $\{M_S\}$ grouped by *month* and *state*

```
select * from view_m_s
order by state asc, month asc;
```

Figure 2.7: Rewritten query for 7. query

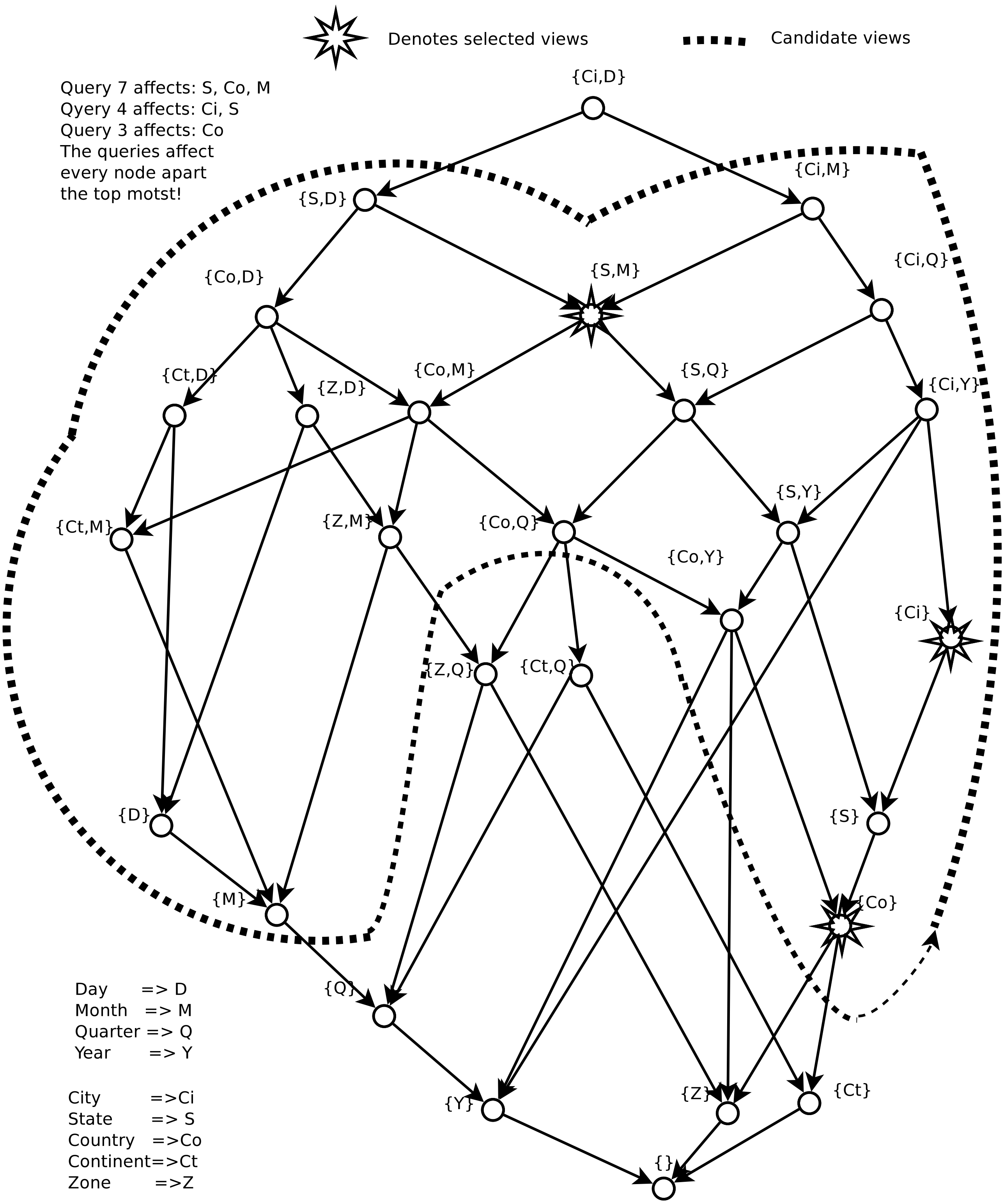


Figure 2.8: Lattice

```
-- 1) determinig size of view or table
select segment_name table, sum(bytes)/(1024*1024) size_MB
from user_extents where segment_type='TABLE'
and segment_name = '&name' group by segment_name;

-- 2) disabling caching 1. method
ALTER SYSTEM FLUSH BUFFER_CACHE;
-- 2) disabling caching 2. method
ALTER TABLESPACE oplatek OFFLINE;

-- 3) enables rewriting the queries by using this views
create materialized view view_co
    build immediate   --not relevant here
    enable query rewrite --this is important
    ...
```

Figure 2.9: Technical details

2.3.2 Indexes

Task

Check what types of indexes does Your DBMS support. Name and describe those of them which are especially useful for speeding up Your queries.

Oracle 10g provides several indexing schemes [5] that provide complementary performance functionality. These are:

- B-tree indexes: the default and the most common
- B-tree cluster indexes: defined specifically for cluster
- Hash cluster indexes: defined specifically for a hash cluster
- Global and local indexes: relate to partitioned tables and indexes
- Reverse key indexes: most useful for Oracle Real Application Clusters applications
- Bitmap indexes: compact; work best for columns with a small set of values
- Function-based indexes: contain the precomputed value of a function/expression
- Domain indexes: specific to an application or cartridge.

A B-tree is a tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time.[6] It is designed to allow quickly finding those tuples that have a specific key value. For this reason, they are suitable for speeding up selection operations.[7]

We will use B-trees to index primary keys and other columns which are used to filter data.

Bitmap indexes are also designed for finding tuples by a specific key but are very structurally different from B-tree indexes. A bitmap index is like two-dimensional boolean arrays of the same size as the number of rows for every value where each element contains a binary value indicating whether the row contains this value. In this structure, the binary value takes up only 1 bit and for each possible value the number of rows bits are used. It is much more space efficient than B-trees if the number of possible values is very low. Another feature of bitmap indexes is the ability to quickly do index intersections.

In our data warehouse we have several columns which contain only a small set of possible values. By using bitmap indexes for these columns, it is possible to reduce the storage space taken up by the indexes. When a query filters data with several of these columns at the same time, the database system will combine these indexes to return data very quickly.

[5] http://docs.oracle.com/cd/B19306_01/server.102/b14231/indexes.htm
[6] http://en.wikipedia.org/wiki/B-tree
[7] Data Warehouse Design: Modern Principles and Methologies, section 11.2.1

Task

For the chosen queries and their versions which use the materialized views, implement indexes to improve their performance (You can try to evaluate several strategies and choose the best one, e.g. best speed/size ratio). Argue Your choice. Describe the overhead (index size, computation time, index update costs, etc.) of implementing the chosen indexes.

The following formulas to calculate the size of indexes are used[8]:

- B-Tree index: $(NK * len(k) + NR * len(r))/(D * u)$ [9]
- Bitmap index: $(NR/D * 8) * NK$

Subscription revenue by country

```
select l.country, sum(price * (1-discount) * quantity) revenue,
  sum(quantity) subscriptions
from sub_subscription s
join sub_location l on s.keylocation = l.keylocation
join sub_date d on s.keydate = d.keydate and d.year = 2010
group by l.country
order by revenue desc
```

There should be B-tree indexes *sub_subscription*(*keyLocation*), *sub_location*(*keyLocation*), *sub_subscription*(*keyDate*), *sub_date*(*keyDate*) to avoid a full table scan when joining two tables. Bitmap indexes are not suitable for this since there are many possible values and such a bitmap index would take up much more space. Assuming 1 million distinct tuples, the B-tree index size would be 21.48 MB (vs 30.52 GB for bitmap indexes). In a B-tree index for every tuple insertion time complexity is $O(log(n))$[10]

There should be an index *sub_date*(*year*) so that it can be used when data is filtered by year. Assuming 365 days per year and 100 years → 36500 tuples. Assuming the total number of distinct values is 100 (covering 100 years), it is reasonable to consider bitmap indexes for this index. A bitmap index would take up 7.129 MB, while a B-tree index would take up 0.07 MB of disk space. While a bitmap index has a 100x larger space overhead, it is negligible on modern systems. Considering the fact that other columns of the table *sub_date* are perfect candidates for bitmap indexes (such as holiday columns: Christmas - yes/no) and the fact that multiple bitmap indexes can be used together to speed up joins, it is reasonable to choose a bitmap index.

Subscription revenue by state and month

```
select l.state, d.month,
  sum(price * (1-discount) * quantity) revenue,
```

[8] Data Warehouse Design: Modern Principles and Methologies, section 11.2.1

[9] Shortcuts used: NK = number of distinct key values, NR = 1000000 number of relations, len(k) = 8 = key length in kilobytes, len(r) = 8 = RID (disk page and location) length in kilobytes, D = 4096 disk page size in kilobytes, u = 1 fill factor for leaves

[10] `http://en.wikipedia.org/wiki/B-tree`

```
  /* avg_revenue_in_this_month_across_all_states */
  avg(sum(price * (1-discount) * quantity))
    over (partition by month) avg_rev
from sub_subscription s
join sub_location l
    on s.keylocation = l.keylocation and l.country = 'Canada'
join sub_date d
    on s.keydate = d.keydate and d.year = 2010
group by l.state, d.month
order by state asc, month asc
```

In addition to in the previously mentioned indexes, there should be an index *sub_location*(*country*) since the *sub_location* table is filtered by the column *country*. Since there is a limited number of countries (in our warehouse around 20 and in total around 194), both B-tree and bitmap indexes can be used. As with the dates discussed previously, the number of total distinct tuples in the $sub_location$ table in the real world would be tens of thousands. While a bitmap index would take up 100x more space than a B-tree index, the size of the index is negligible.

Articles by reads and comments

```
select a.keyarticle id, a.title,
   sum(reads)/sum(comments) comm_by_reads,
   avg(sum(reads)/sum(comments)) over () avg_comm_by_reads,
   sum(reads) reads
from oplatek.artpop_articlepopularity f
join oplatek.artpop_article a on f.keyarticle = a.keyarticle
where a.publicationyear = 2011 and a.publicationmonth = 10
group by a.keyarticle, a.title
order by comm_by_reads - avg_comm_by_reads desc
```

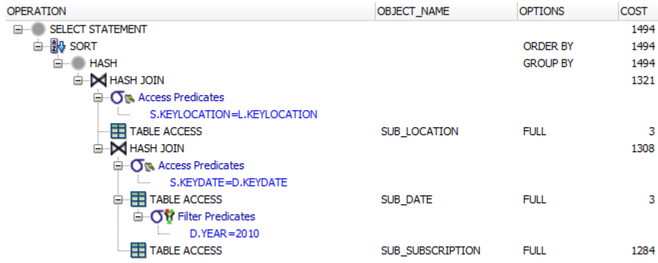
There should be B-tree indexes of *artpop_articlepopularity*(*keyArticle*) and *artpop_article*(*keyArticle*) to speed up the join. Additionally, in theory there should be two different indexes of *artpop_article*(*publicationYear*) and *artpop_article*(*publicationMonth*) or one composite index of *artpop_article*(*publicationYear*, *publica* to speed up filtering articles. Since there can be hundreds of thousands of articles, a bitmap index is not feasible, as it would take a lot of space (assuming 1 million articles, a B-tree index 21.48 MB vs a bitmap index 30.52 GB). Also, there are not many columns in the article popularity dimensions that would be able to advantage of the fast bitmap index intersection.

Task

Measure the queries performance using indexes and compare it to the performance without indexes. Discuss the execution plans.

Subscription revenue by country

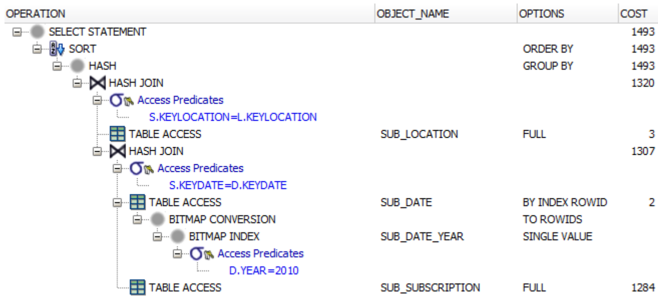
This query takes 0.604s to run without any indexes. Since there are not many rows in the dimension tables (730 date tuples and 240 location tuples), Oracle performs a full table scan on them. A full table scan is also performed on the fact table since all of its data must be aggregated.



OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			1494
SORT		ORDER BY	1494
HASH		GROUP BY	1494
HASH JOIN			1321
Access Predicates			
S.KEYLOCATION=L.KEYLOCATION			
TABLE ACCESS	SUB_LOCATION	FULL	3
HASH JOIN			1308
Access Predicates			
S.KEYDATE=D.KEYDATE			
TABLE ACCESS	SUB_DATE	FULL	3
Filter Predicates			
D.YEAR=2010			
TABLE ACCESS	SUB_SUBSCRIPTION	FULL	1284

After introducing the *sub_subscription*(*keyLocation*) and *sub_subscription*(*keyDate*) indexes (*sub_date*(*keyDate*) and *sub_location*(*keyLocation*) were already primary keys) query time didn't change and the execution plan was the same.

After introducing the *sub_date*(*year*) bitmap index, query time didn't improve but the execution plan shows that it's being used. After seeing no changes, a B-tree index was tried instead, but it didn't have any impact on neither the execution time nor the execution plan.



OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			1493
SORT		ORDER BY	1493
HASH		GROUP BY	1493
HASH JOIN			1320
Access Predicates			
S.KEYLOCATION=L.KEYLOCATION			
TABLE ACCESS	SUB_LOCATION	FULL	3
HASH JOIN			1307
Access Predicates			
S.KEYDATE=D.KEYDATE			
TABLE ACCESS	SUB_DATE	BY INDEX ROWID	2
BITMAP CONVERSION		TO ROWIDS	
BITMAP INDEX	SUB_DATE_YEAR	SINGLE VALUE	
Access Predicates			
D.YEAR=2010			
TABLE ACCESS	SUB_SUBSCRIPTION	FULL	1284

One possible explanation of the reason why Oracle in this case is not using a B-tree index is that Oracle has chosen an execution plan based on costs. Since there are few rows in the dimension tables, a full table scan is fast enough and there is no need to use an index. When this query is run on empty fact & dimension tables, the execution plan shows that indexes would be used as there are is no data from which to estimate costs. It is also worth pointing out that it was observed that other database systems such as MySQL would use the proposed indexes in their execution plans.

It was concluded that introducing indexes for this query didn't bring any improvements due to the fact that there are few rows in the dimension tables. In a real world scenario, there still wouldn't be many rows in these two dimension tables and an index wouldn't improve the performance significantly.

Subscription revenue by state and month

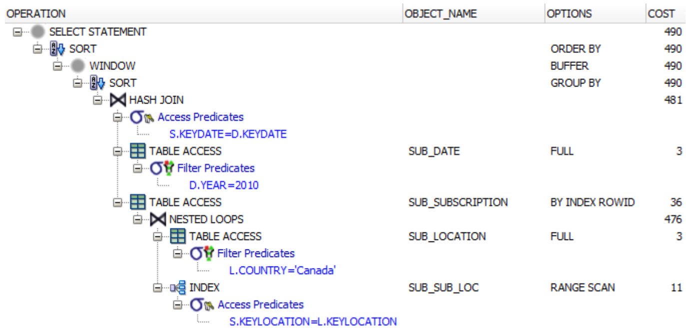
This query takes 0.167s to run without any indexes.

After introducing the *sub_subscription*(*keyLocation*) and *sub_subscription*(*keyDate*) indexes (*sub_date*(*keyDate*) and *sub_location*(*keyLocation*) were already primary

OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			1321
SORT		ORDER BY	1321
WINDOW		BUFFER	1321
SORT		GROUP BY	1321
HASH JOIN			1312
Access Predicates			
S.KEYDATE=D.KEYDATE			
TABLE ACCESS	SUB_DATE	FULL	3
Filter Predicates			
D.YEAR=2010			
HASH JOIN			1308
Access Predicates			
S.KEYLOCATION=L.KEYLOCATION			
TABLE ACCESS	SUB_LOCATION	FULL	3
Filter Predicates			
L.COUNTRY='Canada'			
TABLE ACCESS	SUB_SUBSCRIPTION	FULL	1284

keys) query time improved by 100ms and the query took on average 0.068s to run.

After introducing the *sub_location*(*country*) bitmap index, query time didn't improve but the execution plan shows that it's being used. After seeing no changes, a B-tree index was tried instead, but it didn't have any impact on neither the execution time nor the execution plan.



OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			490
SORT		ORDER BY	490
WINDOW		BUFFER	490
SORT		GROUP BY	490
HASH JOIN			481
Access Predicates			
S.KEYDATE=D.KEYDATE			
TABLE ACCESS	SUB_DATE	FULL	3
Filter Predicates			
D.YEAR=2010			
TABLE ACCESS	SUB_SUBSCRIPTION	BY INDEX ROWID	36
NESTED LOOPS			476
TABLE ACCESS	SUB_LOCATION	FULL	3
Filter Predicates			
L.COUNTRY='Canada'			
INDEX	SUB_SUB_LOC	RANGE SCAN	11
Access Predicates			
S.KEYLOCATION=L.KEYLOCATION			

It was concluded that only the *sub_subscription*(*keyLocation*) index was responsible for reducing the query time by 100ms.

Articles by reads and comments

The query takes about 1 second to run mainly due to the fact that all articles are returned and selected in order to do an aggregation over all of the articles.

The proposed indexes didn't improve the execution time of the query. Different combinations of indexes on *article*'s $publicationYear$ and $publicationDate$ were tried but yielded no performance improvement. Even adding indexes on all involved columns as well as combinations of them had no impact on the running time of the query.

In conclusion, indexing doesn't seem to have any noticable effect on this query due to the fact that it's necessary to scan all rows in order to calculate the average value. The only way to improve this query would be to rewrite it by removing the windowing function.