School of Engineering and Computer Science

SWEN 432
Advanced Database Design and Implementation

Assignment 5

Due date: Sunday 11 June at 23:59 pm

The objective of this assignment is to test your understanding of data mart, materialized views, OLAP specific queries, and your ability to apply this knowledge in the environment of a traditional SQL Server. The Assignment is worth 5.0% of your final grade. The Assignment is marked out of 100.

Question 1. A Data Mart in the PostgreSQL Environment [12 marks]

To accomplish this project you will use PostgreSQL Database Management System. You will reach PostgreSQL Manual from Technical web page that is referenced from SWEN 432 web page. There is also a short instruction how to use PostgreSQL at the end of this assignment.

Short description of the problem

The file

`BookOrdersDatabaseDump.sql`

contains the dump of an operational database that keeps records about customers, books and orders. The description of the database schema is given in Table 1. In the text that follows, this database will be referred as *"Book Orders Database"*.

Besides functional dependencies implied by relation schema keys the *"Book Orders Database"* should also satisfy the following functional dependencies:

City→*District*, and
District→*Country*.

Attribute	Data Type	M. Length	Null	Default	Constraint
Table name: *Customer*, Primary Key: *CustomerID*					
CustomerId	int	4	N		> 0
L_Name	char	15	N		
F_Name	char	15	N		
City	char	15	N		
District	char	15	N		
Country	char	15	N		
Table name: *Cust_Order*, Primary Key: *OrderID*					
OrderId	int	4	N		> 0
OrderDate	date		N		
CustomerId	int	4	N		fk, > 0
Table name: *Order_Detail*, Primary Key: (*OrderID* + *Item_No*)					
OrderId	int	4	N		fk, > 0
ItemNo	int	2	N		
ISBN	int	4	N		fk, > 0
Quantity	int	2	N	1	
Table name: *Book*, Primary Key: *ISBN*					
ISBN	int	4	N		> 0
Title	char	60	N		
Edition_No	int	2	Y	1	> 0
Price	decimal	6, 2	N		> 0
Table name: *Author*, Primary Key: *AuthorId*					
AuthorId	int	4	N		> 0
Name	char	15	Y		
Surname	char	15	N		
Table name: *Book_Author*, Primary Key: (*ISBN* + *AuthorId*)					
ISBN	int	4	N		fk, > 0
AuthorId	int	4	N		fk, > 0
AuthorSeq_No	int	2	Y	1	

Table 1.

You are asked to create and populate your own operational “Book Orders Database” using the corresponding dump file. You are also asked to create your Data Mart according to the description of its schema in Table 2, and to populate it by extracting, transforming, and loading data from the operational “Book Orders Database”. Use PostgreSQL commands and functions to accomplish the tasks.

When building Time dimension, note that:

- TimeId should be a sequence generated by PostgreSQL. That sequence is associated with the Cust_Order.OrderDate values in an ascending manner (the earliest Cust_Order.OrderDate date is associated with the TimeId =1).
- There are various PostgreSQL functions that allow automatic transformation of Cust_Order.OrderDate values into appropriate surrogates of DayOfWeek, Month, and real Year values.
- The DayOfWeek attribute takes values from the set {'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday')}.
- The Month attribute takes values from the set {'January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December'}.

Attribute	Data Type	M. Length	Null	Default	Constraint
Dimension name: *Customer*, Primary Key: *CustomerID*					
CustomerId	int	4	N		> 0
L_Name	char	15	N		
F_Name	char	15	N		
City	char	15	N		
District	char	15	N		
Country	char	15	N		
Dimension name: *Time*, Primary Key: *TimeId*					
TimeId	int	4	N		
OrderDate	date		N		
DayOfWeek	char	10	N		
Month	char	10	N		
Year	int	4	N		
Dimension name: *Book*, Primary Key: *ISBN*					
ISBN	int	4	N		> 0
Title	char	60	N		
Edition_No	int	2	Y	1	> 0
Price	decimal	6, 2	N		> 0
Fact table name: *Sales*, Primary Key: (*CustomerId* + *TimeId* + *ISBN*)					
CustomerId	int	4	N		fk > 0
TimeId	int	4	N		fk > 0
ISBN	int	4	N		fk > 0
Amnt	decimal	6, 2	N		> 0

Table 2.

When building Sales Fact table, note that

Amnt = SUM(Order_Detail.Quantity * Book.Price).

You may use Customer and Book tables of your *“Book Orders Database”* as Data Mart dimension tables. Even more, you may find it convenient to implement your Time dimension table and Sales Fact table within your *“Book Orders Database”*, and hence to have just one physical database.

Question 2. Aggregate Queries [14 marks]

Pavle was asked to find the average amount of money spent by all customers on buying books for all days so far. Since there already existed the following materialized view:

```
create materialized view avg_amnt_view as select customerid,
avg(amnt) as avg_amnt from sales group by customerid;
```

Pavle decided to use the materialized view and avoid computing aggregate from the scratch. Here is what he has done and what he has got as the result:

```
select avg(avg_amnt) from avg_amnt_view;
         avg
----------------------
 219.6439318342364916
(1 row)
```

To satisfy the poor curiosity, Pavle also ran the following query:

```
select avg(amnt) from sales;
         avg
----------------------
 165.8689839572192513
(1 row)
```

The result obtained was completely different. Pavle spent hours and days looking for the mistake in vain. Then he decided to ask you for help. Tell Pavle which result was correct? What Pavle did wrong?

Note: Pavle used the old version of the data mart to calculate averages. You will get different results using the new version.

Question 3. OLAP Queries [20 marks]

a) **[5 marks]** Use SQL to retrieve from your Data Mart: customer id’s, names and surnames of five customers who spent the largest amount of money buying books. This query uses two OLAP specific operations, name them.

b) **[15 marks]** Use SQL to find from your Data Mart and the operational database whether the customer who spent the greatest amount of money buying books did this by issuing many orders with smaller amounts or a few orders with greater amounts of money, or even great number of orders with greater amounts of money. Base your answer on an appropriate average value and the percentage of best buyer's orders being smaller or greater than this average. What is the name of that OLAP specific operation? (You may use a stepwise procedure to solve the question.)

Hint:
Let:

- `ord_avg_amnt` be the average amount of money of all orders,
- `no_of_ord` be the number of orders issued by the customer who spent the greatest amount of money buying books (the best buyer), and
- `perc_of_ord` be the percentage of orders issued by the best buyer that had a greater total amount than the `ord_avg_amnt`.

If:

- `perc_of_ord` is 75% or greater, we estimate that the best buyer has issued a greater (than average) number of orders with greater (than average) amounts of money,
- `perc_of_ord` is between 50% and 75%, we estimate that the best buyer has issued a greater (than average) to medium number of orders with greater (than average) amounts of money,
- `perc_of_ord` is between 25% and 50%, we estimate that the best buyer has issued a small to medium number of orders with greater (than average) amounts of money,
- `perc_of_ord` is between 0% and 25%, we estimate that the best buyer has issued a small number of orders with greater (than average) amounts of money.

Question 4. Queries Against Materialized Views [24 marks]

a) [12 marks] Use SQL to materialize the following two views:

```
CREATE MATERIALIZED VIEW View1 AS
SELECT c.CustomerId, F_Name, L_Name, District, TimeId,
DayOfWeek, ISBN, Amnt
FROM Sales NATURAL JOIN Customer c NATURAL JOIN Time;

CREATE MATERIALIZED VIEW View2 AS
SELECT c.CustomerId, F_Name, L_Name, Year, SUM(Amnt)
FROM Sales NATURAL JOIN Customer c NATURAL JOIN Time
GROUP BY c.CustomerId, F_Name, L_Name, Year;
```

Use PostgreSQL `EXPLAIN ANALIZE` (and `VACUUM ANALYZE`) command to get time needed to retrieve five best buyers of question 3.a) when the SQL statement is issued against:

1. The *"Book Orders Database"*

2. The Data Mart,
3. The view View1, and
4. The view View2.

Explain your findings.

b) [12 marks] Use SQL to materialize the following view:

```
CREATE MATERILIAZED VIEW View3 AS
SELECT District, TimeId, DayOfWeek, ISBN, SUM(Amnt)
FROM Sales NATURAL JOIN Customer NATURAL JOIN Time_Dim
GROUP BY District, TimeId, DayOfWeek, ISBN;
```

Use PostgreSQL `EXPLAIN ANLYZE` (and `VACUUM ANALYZE`) command to get time needed to retrieve the country whose inhabitants spent the largest amount of money buying books when the SQL statement is issued against:

1. The *“Book Orders Database”*
2. The Data Mart,
3. The view View2, and
4. The view View3.

Explain your findings.

Question 5. Queries with WINDOW Function [30 marks]

To answer the following two questions you will need to apply `WINDOW` function onto your datamart. Read PostgreSQL manual to find out more how PostgreSQL supports the `WINDOW` function.

a) [15 marks] Business analysts want to contrast the sum of ammounts spent by customers buying books in April and May 2017 with the average amount spent by all customers from a city. So, in your answer, customers, represented by their `customerId`’s and first names, need to be grouped by `cities`. Also, place the average after the sum of ammount column.

b) [15 marks] Business analysts want to contrast the daily sums of ammounts spent by all customers from a city buying books in April and May 2017 with the cumulative sum from the start of April including the current day. In the query result, you need to display the following columns in the following order: `city`, `timeid`, `day`, `sum(amnt)`, `cumulative_sum`.

What to hand in:

- All answers both electronically and as a hard copy.
- A statement of any assumptions you have made.
- Answers to the questions above, together with the listing and the result of each query. In your answers, copy your SQL command, and PostgreSQL

message to it from console pane. Do not submit contents of any tables.

Using PostgreSQL on a Workstation

We have a command line interface to PostgreSQL server, so you need to run it from a Unix prompt in a shell window. To enable the various applications required, first type

```
> need postgresql
```

You may wish to add the "`need postgresql`" command to your `.cshrc` file so that it is run automatically. Add this command after the command need `SYSfirst`, which has to be the first need command in your `.cshrc` file.

There are several commands you can type at the unix prompt:

```
> createdb  ⟨db name⟩
```

Creates an empty database. The database is stored in the same PostgreSQL default school cluster used by all the students in the class. To ensure security, you must name your database by your **`userid`**. You only need to do this once (unless you get rid of your database to start again).

```
> psql  [ –d  ⟨db name⟩ ]
```

Starts an interactive SQL session with PostgreSQL to create, update, and query tables in the database. The `db name` is optional (unless you have multiple databases)

```
> dropdb  ⟨db name⟩
```

Gets rid of a database. (In order to start again, you will need to create a database again)

```
> pg_dump  -i  ⟨db name⟩ > ⟨file name⟩
```

Dumps your database into a file in a form consisting of a set of SQL commands that would reconstruct the database if you loaded that file.

```
> psql –d <database_name> -f <file_name>
```

Copies the file `<file_name>` into your database `<database_name>`.

Inside an interactive SQL session, you can type SQL commands. You can type the command on multiple lines (note how the prompt changes on a continuation line). End commands with a '`;`'

There are also many single line PostgreSQL commands starting with '`\`' . No '`;`' is required. The most useful are

`\?` to list the commands,

`\i` `⟨file name⟩`
loads the commands from a file (eg, a file of your table definitions or the file of

data we provide).

`\dt` to list your tables.

`\d` ⟨`table name`⟩ to describe a table.

`\q` to quit the interpreter

`\`**`copy`** `<table_name>` **`to`** `<file_name>`
Copy your `table_name` data into the file `file_name`.

`\`**`copy`** `<table_name>` **`from`** `<file_name>`
Copy data from the file `file_name` into your table `table_name`.

Note also that the PostgreSQL interpreter has some line editing facilities, including up and down arrow to repeat previous commands.
For longer commands, it is safer (and faster) to type your commands in an editor, then paste them into the interpreter!