School of Engineering and Computer Science

SWEN 432
Advanced Database Design and Implementation

Assignment 5

This Model Answer is produced using an older version of Data Mart. So, procedures and SQL statements are correct, but the numerical results are different to what you have achieved using the new version of Data Mart.

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.

Include all numbers of rows inserted in your answer.

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).
```

The meaning of the attribute `Amnt` is: the amount of money a customer spent buying a book on a date (`TimeId`).

Also note, a customer may issue more than one order on the same date.

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.

ANSWER

```
% create book_orders_17
% psql –d book_orders_17 –f BookOrdersDatabaseDump_17.sql

book_orders_17=> create table time_dim (timeid int primary
key, orderdate date not null, dayofweek char(10) not null,
month char(10) not null, year smallint not null);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit
index 'time_dim_pkey' for table 'time_dim'

book_orders_17 => create sequence time_seq;

book_orders_17=> insert into time_dim(select
nextval('time_seq'), orderdate, to_char(orderdate,'Day')
as dayofweek, to_char(orderdate, 'Month') as month,
extract (year from orderdate) from (select distinct
orderdate from cust_order order by orderdate) as o,
time_seq);
INSERT 0 107
Note: time_seq is a table having just one row. That row
contains current sequence number.

book_orders_17=> create table sales (timeid int not null
references time_dim, customerid int not null references
customer, isbn int not null references book, amnt
decimal(6,2) default 0, primary key (timeid, customerid,
isbn) );
CREATE TABLE

insert into sales select t.timeid, c.customerid, b.isbn,
sum(quantity*price) from time_dim t natural join
customer c natural join book b natural join cust_order o
natural join order_detail d group by t.timeid,
c.customerid, b.isbn;
INSERT 0 798
```

Question 2. Aggregate Queries [14 marks]

Pavle was asked to find the average amount of money spent by all customers all buying books on all days so far (note: “all” implies the top of an attribute hierarchy and leads to dropping a whole dimension). The request may also be refrased in the following way: average amount of money spent by a customer, buying a book, and a date. 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.

ANSWER

The second answer is correct, since the first answer computes an average of an average, and an average of an average is not an average.

Justification:

```
book_orders_17=> create materialized view avg_amnt_view as
select customerid, avg(amnt) as avg_amnt from customer
natural join sales group by customerid;
SELECT 79

book_orders_17=> select avg(avg_amnt) from avg_amnt_view;
         avg
----------------------
 213.6790090091575832
(1 row)

book_orders_17=> select avg(amnt) from sales;
         avg
----------------------
 167.2117794486215539
(1 row)

//We can't use the materialized view avg_amnt_view to
compute a courser grain average, since the average function
is algebraic and the view does not contain a p-tuple.
Accordingly, the view does not satisfy the aggregate
computability test for query rewriting.

//To use a materialized view for computing a courser grade
average, we need the ability to recreate the sum of amounts
spent by each customer and count of purchases done by
customers.

//To check claims above, we make the following view:

book_orders_17=> create materialized view
avg_amnt_count_view as select customerid, avg(amnt) as
avg_amnt, count(*) as group_count from customer natural
join sales group by customerid;
SELECT 79

book_orders_17=> select (select sum(avg_amnt*group_count)
from avg_amnt_count_view)/(select sum(group_count) from
avg_amnt_count_view) as total_avg;
      total_avg
----------------------
 167.2117794486215539
(1 row)
```

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.

ANSWER

```
select c.customerid, f_name, l_name, sum(amnt) as tot_amnt
from customer c natural join sales s
group by c.customerid, f_name, l_name order by tot_amnt
desc limit 5;
  customerid |    f_name     |    l_name     | tot_amnt
-------------+--------------+--------------+----------
           1 | Kirk         | Jacson       | 17810.00
           3 | Peter        | Andree       | 14100.00
          14 | Craig        | Anslow       | 11780.00
           2 | May-N        | Leow         |  7145.00
          79 | Jiajun       | Liang        |  6095.00
(5 rows)
OLAP operations: Roll-Up and TopN
```

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 great 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.

ANSWER

Since the data mart does not contain orders, we need to do **drill down** and to query the operational database.

PROCEDURE

Step 1 - best_buyer

```
book_orders_17=> create view best_buyer as select
c.customerid, f_name, l_name, sum(quantity*price) as
tot_amnt from customer c natural join cust_order o natural
join order_detail d natural join book b group by
c.customerid, f_name, l_name order by tot_amnt desc limit
1;
CREATE VIEW
```

Step 2 - average money spent on orders

```
book_orders_17=> create view ord_avg_amnt as select (select
sum(price*quantity) from cust_order natural join
order_detail natural join book)/(select count(*) from
cust_order) as avg_per_order;

CREATE VIEW
```

Step 3 - number of orders made by best buyer

```
book_orders_17=> create view no_of_ord as select count(*)
from cust_order where customerid = 1;
CREATE VIEW
```

Step 4 - orders having total greater than average

```
create view greater_total_orders as select orderid,
sum(price*quantity) from cust_order natural join
order_detail natural join book where customerid = 1 group
by orderid having sum(price*quantity) > (select * from
ord_avg_amnt);
```

//Step 5 - find the percentage

```
book_orders_17=> select (100*(select count(*) from
greater_total_orders) /(select * from no_of_ord)) || '%' as
perc_of_ord;
 perc_of_ord
-------------
 71%
(1 row)
```

So, we estimate Kirk has issued a great to medium number of orders with amounts greater than average.

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.

ANSWER

Operational Database:

```
book_orders_17=> explain analyze select c.customerid,
f_name, l_name, sum(quantity*price) as tot_amnt
from customer c natural join cust_order o natural join
order_detail d natural join book b group by c.customerid,
f_name, l_name order by tot_amnt desc limit 5;
 Planning time: 0.532 ms
 Execution time: 4.131 ms
(24 rows)
```

Data Mart:

```
book_orders_17=> explain analyze select c.customerid,
f_name, l_name, sum(amnt) as tot_amnt from customer c
natural join sales s group by c.customerid, f_name, l_name
order by tot_amnt desc limit 5;
 Planning time: 0.222 ms
 Execution time: 2.520 ms
(14 rows)
```

View1:
```
book_orders_17=> explain analyze select customerid, f_name,
l_name, sum(amnt) as tot_amnt from view1 group by
customerid, f_name, l_name order by tot_amnt desc limit 5;
 Planning time: 0.107 ms
 Execution time: 1.465 ms
(9 rows)
```

View 2:
```
book_orders_17=> explain analyze select customerid, f_name,
l_name, sum(sum) as tot_amnt from view2 group by
customerid, f_name, l_name order by tot_amnt desc limit 5;
 Planning time: 0.145 ms
 Execution time: 0.356 ms
(9 rows)
```

Analysis
The rang list:
```
view 2          plan_time: = 0.145 ms, exe_time: 0.356 ms
view 1          plan_time: = 0.107 ms, exe_time: 1.465 ms
data mart       plan_time: = 0.222 ms, exe_time: 2.520 ms
operational db  plan_time: = 0.532 ms, exe_time: 4.131 ms
```

The query against the operational database takes longest, since it performs three joins and a grouping.

The query against the data mart is the second longest, since it performs two joins and a grouping.

The query against the view 1 is the second best, it does not perform any joins, but performs grouping from the scratch that uses sorting.

The query against view 2 is the best, since it performs no joins, and no grouping. All of these are pre computed.

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.

ANSWER

Operational Database

```
explain analyze select country, sum(price*quantity) as
tot_country from customer natural join cust_order natural
join order_detail natural join book group by country order
by tot_country desc limit 1;
 Planning time: 0.472 ms
 Execution time: 3.787 ms
(24 rows)
```

Data Mart:

```
explain analyze select country, sum(amnt) as tot_country
from customer natural join sales group by country order by
tot_country desc limit 1;
 Planning time: 0.199 ms
 Execution time: 2.161 ms
(14 rows)
```

View 2:

```
explain analyze select country, sum(sum) as tot_country
from view2 natural join customer group by country order by
tot_country desc limit 1;
 Planning time: 0.345 ms
 Execution time: 0.540 ms
(14 rows)
```

View 3

```
explain analyze select country, sum(sum) as tot_country
from view3 natural join (select distinct district, country
from customer) as c group by country order by tot_country
desc limit 1;
 Planning time: 0.184 ms
 Execution time: 2.213 ms
(17 rows)
```

Analysis

The rang list:

```
view 2         plan_time: = 0.345 ms, exe_time: 0.540 ms
data mart      plan_time: = 0.199 ms, exe_time: 2.161 ms
view 3         plan_time: = 0.184 ms, exe_time: 2.213 ms
operational db plan_time: = 0.472 ms, exe_time: 3.787 ms
```

The query against the view 2 is the fastest. It performs one join, but is able to utilize the precomputed aggregate, since sum is a distributive aggregate function.

The query against the data mart is the second best. It performs only one join as the query against the view 2, but has to compute the aggregate from the scratch.

The query against the view 3 is the third. It performs also only one join and is able to utilize the precomputed aggregate, but the join requires the execution of the distinct clause. The distinct clause is implemented using sort and its time complexity is close to the complexity of a join.

The query against the operational database has the worst performance. It executes three joins and computes the aggregate from the scratch.

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.

ANSWER

```
select city, customerid, f_name, sum_amnt, avg(sum_amnt)
over (partition by city) from (select city, customerid,
f_name, sum(amnt) as sum_amnt from sales natural join
customer natural join time_dim where year = 2017 and month
in ('April', 'May') group by city, customerid, f_name) as
t;
    city    | customerid |     f_name     | sum_amnt |  avg
------------+------------+----------------+----------+-------------------
 Beijing    |         87 | Liu            |   920.00 | 557.50
 Beijing    |         82 | XiaoHan        |   195.00 | 557.50
 Shanghai   |         86 | Linfeng        |   635.00 | 635.00
 Upper Hutt |         84 | Ryian          |   670.00 | 670.00
 Wellington |         85 | Lucy           |   755.00 | 728.89
 Wellington |         89 | Amon           |    75.00 | 728.89
 Wellington |         92 | Ben            |   315.00 | 728.89
 Wellington |          1 | Kirk           |   700.00 | 728.89
 Wellington |         91 | Bradley        |  1175.00 | 728.89
 Wellington |         83 | Alex           |   725.00 | 728.89
 Wellington |         88 | Ewan           |   975.00 | 728.89
 Wellington |         90 | Venkata        |   690.00 | 728.89
 Wellington |         93 | Dany           |  1150.00 | 728.89
(13 rows)
```

b) [12 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`.

ANSWER

```
select city, timeid, dayofweek, sum_amnt, sum(sum_amnt)
over w from (select city, timeid, dayofweek, sum(amnt) as
sum_amnt from sales natural join customer natural join
time_dim where year = 2017 and month in ('April', 'May')
group by city, timeid, dayofweek) as t WINDOW w AS
(partition by city order by timeid rows between unbounded
preceding and current row);
      city       | timeid | dayofweek  | sum_amnt |   sum
-----------------+--------+------------+----------+--------
 Beijing         |     96 | Sunday     |   405.00 |  405.00
 Beijing         |     97 | Monday     |   515.00 |  920.00
 Beijing         |    103 | Friday     |   195.00 | 1115.00
 Shanghai        |     95 | Saturday   |   635.00 |  635.00
 Upper Hutt      |     92 | Wednesday  |   570.00 |  570.00
 Upper Hutt      |     93 | Thursday   |   100.00 |  670.00
 Wellington      |     91 | Tuesday    |   440.00 |  440.00
 Wellington      |     94 | Friday     |   755.00 | 1195.00
 Wellington      |     98 | Tuesday    |   765.00 | 1960.00
 Wellington      |     99 | Wednesday  |   210.00 | 2170.00
 Wellington      |    100 | Thursday   |    75.00 | 2245.00
 Wellington      |    101 | Friday     |  1110.00 | 3355.00
 Wellington      |    102 | Saturday   |   305.00 | 3660.00
 Wellington      |    104 | Saturday   |   285.00 | 3945.00
 Wellington      |    105 | Thursday   |   515.00 | 4460.00
 Wellington      |    106 | Friday     |  1000.00 | 5460.00
 Wellington      |    107 | Sunday     |  1100.00 | 6560.00
(17 rows)
```

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, just the number of rows inserted.

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!