Assignment 2, Q4 report

Submitted by

Azmyin Md. Kamal, C00441440

Date: 4/8/2020

List of software used

1. XAMPP Server (contains APACHE + MYSQL servers for ease of database management)
2. Python 3.8: Primary language for parsing user inputs to SQL queries
3. Visual Studio's Code 2020: Primary IDE
4. Python-MySQL module: Libraries to passing python variables as SQL queries back to the MySQL server running on XAMPP

For ease of explaining my code implementation, the following data mart was implemented programmatically. Please note that, though this datamart is very simplistic, my code is scalable and in theory is capable of creating a star schema with 4 or more dimensions. However, due to shortage of time, this much extensive testing was not possible. Additionally, the Rollup script is exclusive to the datamart shown in Figure 1 the reasons for which is explained in section d).

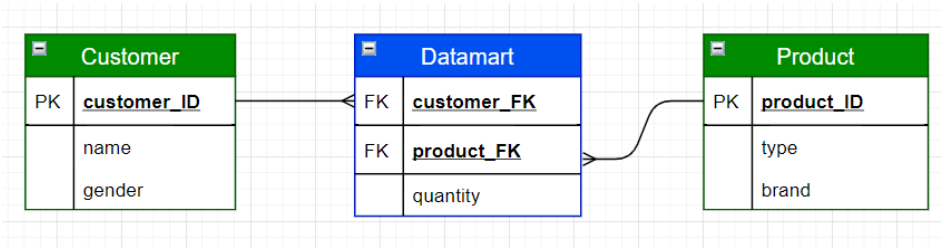

Figure 1: Datamart schema (correct this)

a) Task 1: Ask user to specify the dimensions and attributes: For this task, I have written a sequence of python statements in the file "**a_dim_attr.py**" where I utilize python's *input()* function to query the user for number of dimensions. This number is utilized to run a loop for a chosen amount of times within which the user programmatically enters the name followed by the number of attributes and, finally the names of the attributes themselves. Note with each execution of this loop a SQL query to create a table with the chosen attribute is passed to the MySQL server.

Sample code:

```
sql=(f"CREATE TABLE {dim_name} ({dim_name}_ID INT NOT NULL PRIMARY KEY AUTO_IN
CREMENT)")
cursor.execute(sql)
```

Also note that, *cursor.execute()* is a python-mysql method that transfers the parsed SQL query based on user input to the MySQL database. This function is also used to read data back from the server. I have also employed the try – except program guards to test for bugs and a checking function which ensure that the user does not accidentally input string in place of integers or vice versa during the input phase.

Using the INSERT dynamic webpage in the "phpMyAdmin" page hosted automatically by the XAMPP server, we can insert data to each of the dimension table at this stage. However, I opted to not insert data at this stage in lie of advancing to section c) where the datamart generated automatically with proper

referential constraints. On a concluding note, within this script, a loop collects and records the name of the dimension tables specified by the user in text file called "dimension_name.txt". This txt file is a crucial component for automating all the steps in section c).

b) Task 2: Define a concept hierarchy: A script called "**b_concept.py**" was written, where the user is asked to define a "name" for the table that will store the hierarchy information. The user is presented with a series of instructions to populate the level information for each of the created dimension since the concept level table has only two columns, "dim_name" and "level". For each of the dimensions created in Q 4a), the user is required to re-enter the names and attributes in the following format
"dimension == lvl0 << lvl1 << lvl2 << .... << lvlM"
Example: location == city << street << country << state
Note this implementation is a manual way of re-entering information already entered by user once in Q 4a). Unfortunately, I do know of a way to automate this stage but due to shortage of time, I have intentionally used the manual method to devote more time to Q4 c) and Q4 d). Figure 2 shows the hierarchy table after running the "**b_concept.py**" script in phpMyAdmin website.

Figure 2. Generated concept hierarchy table

c) Task 3: Automatic generation of the Datamart: I have separated the solution for this part into two separate scripts which must be executed sequentially. In script "**c1_ask_data_mart.py**", the user is asked to enter the name of the datamart to be created. This string is saved in a text file. In script "**c2_create_data_mart.py**", the required datamart is created sequentially in the following steps

i. Connect to the "datamart" by reading the name from a txt file called "datamart_name.txt"
ii. Ask user for the name of the fact table
iii. Create fact table with a "dummy" column. More on this later
iv. Ask user to define the "measure" attributes
v. Create columns defined by user as "measure" attributes
vi. Drop the dummy column
vii. Based on the names present in the "dimension_name.txt" file, import those dimension tables having the same name, same primary key id and attributes.
viii. Generate foreign key columns equal to the number of total primary keys (from each dimensions) present in the fact table. For instance, if number of dimensions is two (customer and product) then in the fact table "datamart" (Refer to Figure 1), we will generate two columns with the names "customer_FK" and "product_FK".
ix. Add corresponding foreign key relationships
x. Finally, generate a composite primary key in combinations of the foreign keys present in the fact table.

Figure 3 shows the table structure for the "stores" fact table within the "datamart" whilst Figure 4 shows the validation of the above steps with a dynamically generated star schema.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	customer_FK	int(11)			No	*None*			Change Drop More
2	product_FK	int(11)			No	*None*			Change Drop More
3	quantity	int(6)			Yes	*NULL*			Change Drop More

Figure 3: "stores" fact table

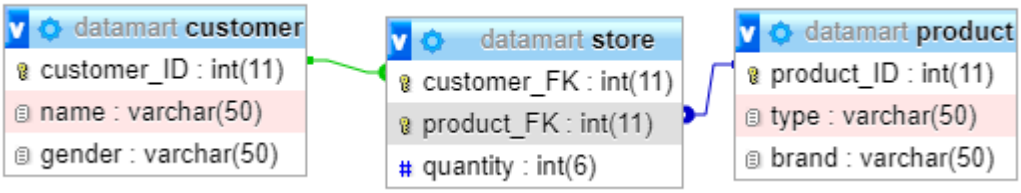

Figure 4: Generated Star Schema

Please note that, Figure 4 was not generated the way it has been presented in this document. While phpMyAdmin does provide a dynamic webpage to view the schema, I had to manually position all the elements in their appropriate positions to resemble the star schema structure. After verifying that the datamart was setup properly I utilized the builtin INSERT tool to add the following data to their respective tables

customer		
ID	name	gender
200	Azmyin	M
201	Sumayia	F

product		
ID	type	brand
100	Laptop	Asus
101	Console	Xbox Series X
102	Console	PS5

stores		
customer_FK	**product_FK**	**quantity**
201	101	5
201	100	10
201	102	5
200	100	10
200	101	20
200	102	5

d) Task 4: Rollup and Drill-down operation: I was only able to implement the Roll-up operation since the generated cuboid is a base cuboid. Figure 5 shows the output of the script "**d_rollup.py**" which has been formed by a combination of the following SQL query implemented twice, once for grouping by customers and second for products.

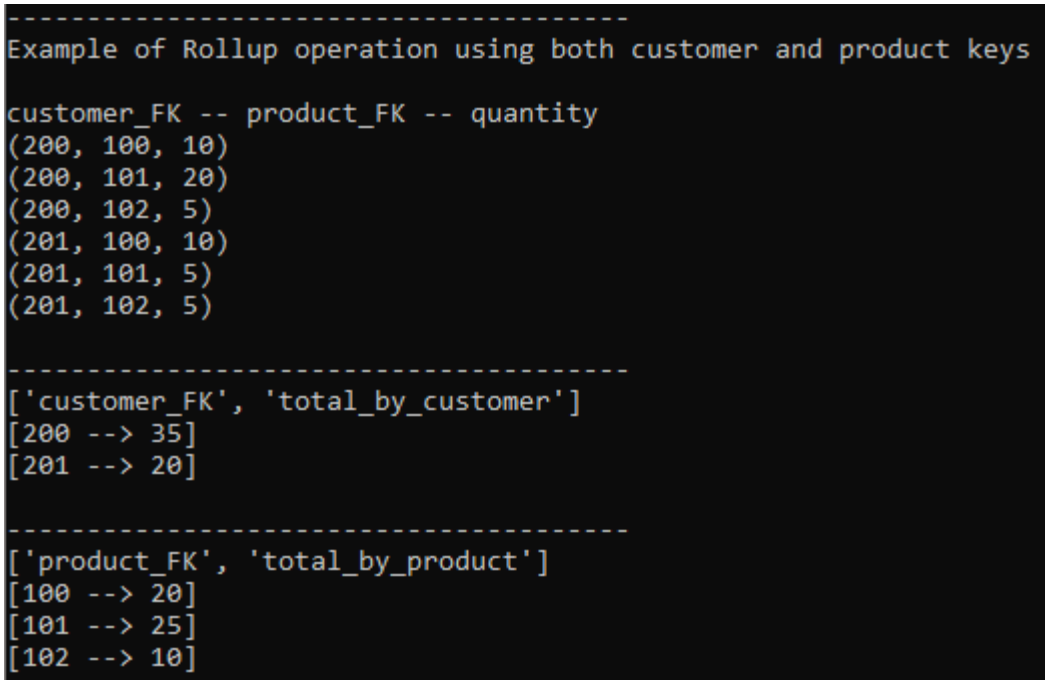

Figure 5: Rollup operation

sql = ("SELECT customer_FK, SUM(quantity) AS total_by_customer FROM store GROUP BY custom
er_FK")

A big drawback of this script is that, it IS specific for this particular data mart. Due to shortage of time and investigating various errors that cropped throughout the development time, I was unable to generalize this script. However, the framework for generalizing this script is same as that of "**c2_create_data_mart.py**" where the key idea is to import important variable names from an external text files so that we are not required to run different scripts at the same time and introduce redundant user interactions with the software.

Note that, for both the aggregates, the total computes to 55 units which is equivalent to the total number of electronic items purchased by the two customers bearing IDs 200 and 201 respectively. For additional verification, both the customers purchased a total of 20 laptops (product_FK = 100) which is same as the result shown in Figure 5.