Problem description

Imagine your client is an auto parts retailer (with 80 locations in the U.S. and Canada, and they sell performance auto parts.

When customers walk into a client store to purchase an item, if the item is in store they can be rung up at the register there. If the item needs to be ordered, the store employee takes the customers information, orders the part, and a 1-3 weeks later the items is shipped to the store and the customer comes in to purchase it. The item is then rung up at that point.

The client stores this purchase data in two tables of their database. The first table, ClosedOrders, stores the information from the items purchased at the store. The OpenOrders table stores the information about items that are ordered to be picked up later. After the item is picked up, the record is deleted from OpenOrders and a new record is created in ClosedOrders to capture the sold item.

There are the fields in each table:

ClosedOrders:

invoice_id – *the unique id for each transaction*

line_id– *a unique id for each item sold with each transaction (ie, multiple items can be purchased with each invoice)*

store_id– *integer, store the transaction took place in*

time_stamp – *The date and time of the purchase, all in the same time zone across stores*

product_id – *The unique id for the product sold within the transaction (every product that the client sells has a unique product_id)*

quantity_sold – *The number of the product sold with the transaction (ie, you could purchase 2 of the same item)*

sales – *the sales received by the client for the item sold*

cost – *the cost of the item sold for the client (ie, not the cost to the customer, but the cost to the client for the product)*

OpenOrders:

invoice_id – *the unique id for each order*

line_id– *a unique id for each item ordered with each transaction (ie, multiple items can be ordered with each invoice)*

store_id– *integer, store the order took place in*

time_stamp – *The date and time of the order, all in the same time zone across stores*

product_id – *The unique id for the product ordered within the transaction*

quantity_sold – *The number of the product ordered (ie, you could purchase 2 of the same item)* sales – *the sales associated with the sold (ie the sales the client will receive when the customer picks up and purchases the item)*

cost – *the cost of the item sold for the client (ie, not the cost to the customer, but the cost to the client for the product)*

Note that the structure of the ClosedOrders and OpenOrders tables are the same with the only difference that OpenOrders represent items ordered rather than purchased.

There is a third table ProductCatalog that has every product_id in it and a column called category_id that indicates what category the item belongs to. As an example, a specific product (e.g., Penzoil 20W-50 Conventional Motor Oil-5 qt) belongs to one category, in this instance "Motor Oil" would be the category. Every product is associated with one and only one category_id. Note that the "product_id" field in this table is the same as the "product_id" field in the ClosedOrders and OpenOrders tables.

With the three tables described above, you are going to write several SQL queries to create tables we can use to analyze the clients transaction data.

Questions

Below are a series of questions asking how you would execute SQL to query the three tables described above. Feel free to use whatever variant of the SQL language you are most comfortable with (MySQL, PostgreSQL, Oracle SQL, etc.), but just note what that is in the first question. If you have to make assumptions about the table structures or values to answer any of the following questions, please just note those assumptions in your answers. Please put your SQL answers and any assumptions below each question.

Question 1

With the ClosedOrders table, create a table that summarizes the total sales, total quantity, total profit per store. Note that profit can be calculated as sales - cost.

Question 2

Building on your query above, create a similar table that now summarizes sales, quantity, and profit per store, per product (ie, sales for each product_id sold in each store_id)

Question 3

Building on your query above, create a similar table that summarizes the same metrics (sales, quantity, profit) per store, per product, per day (ie, products sold each day, for each store)

Question 4

Building on your query above, change the time interval from day to week – ie, summarize sales per store, per product, per week (instead of per day). For extra credit, make the week start on Tuesdays rather than the SQL default grouping.

Question 5

Building on your query above, use the ProductCatalog table to summarize the same metrics (sales, quantity, profit) by store, by week, and by category_id (rather than by product_id).

Question 6

Very good. Now for the hardest and most important question. First a bit of explanation:

> Using just the ClosedOrders table only gives us half of a picture of what demand there is for each product per day. To truly get a sense of how much people demand each product you really want to get the SUM of open orders and closed orders per store, per week, per product. Doing so will truly tell you how much of each item customers came into the store to by—whether the store had it in stock and sold it then (ClosedOrders) or had to order the item for the customer to come back and get it (OpenOrders).

Given the problem above, I would like you to create a table that gives total sales, total quantity, total profit per store, per week, per product (similar to the above) but for these totals include the sum of OpenOrders and ClosedOrders. Write below the SQL you would use to create this table, and also describe what types of SQL you would use to check that your solution was correct. (Note that there are more than one way to solve this problem.)