Business Strategy, A Consistent Change

Anthony Coots

Western Governors University

Data never lies, at least, it will always tell you what you ask it. In the world of commerce, most businesses share a main goal: profit. Profit can build or fall for many different reasons; however, it is usually a safe bet to make changes based on whether or not product(s) are selling. There is an infinite number of ways to hypothesize, predict, so on and so forth how to make profitable changes within a business. Starting with data.

PART A:

A company may oversee many different stores, each more or less profitable in different areas than others. A question that roots itself in such problem asks, "*what and where can the business improve?*". Using the DVD Database provided, the following suggests the structure of data:

1. A store ID, its location, each category, the sum of total sales for each category per store and an indication of whether or not the category is above the stores average sales per category will make up the detailed table structure while the summary table would consist of the records that are below average for each store containing store ID, category and sum, seeking the most improvement.
    a. DETAILED TABLE:
        i. store_ID, address, category, tot_sales, avg_flag.
    b. SUMMARY TABLE:
        i. store_ID, category, tot_sales.

2. Store ID presumably is of type INTEGER. Each store address uses numbers and letters, calling for VARCHAR representation. Each category a string of characters with no numbers, CHAR is

appropriate, the tot_sales a NUMERIC(6,2) representation, of profit in the single thousands, often a two decimal number and CHAR for the above or below average flag.

3. The necessary tables include the category, film_category, film, inventory, rental, payment, staff, store and address tables.

4. Our avg_flag field will require a translation such that, for each store (not to be mixed with other stores sales), output a ‘below average’ or ‘at or above average’ text result for each row from a greater, equal, or less than calculation.

5. Use of the detailed table could assist in business decisions such as, but not limited to, a local managerial team looking to make more profit, cut out categories from its store, or setting up community polls on improving selections for which categories are underwhelming. The summary table would be used on a national level, for commercial research being reported from each district and relevant store without overwhelming corporate level positions of data that may not be necessary at the time.

6. This report would be refreshed yearly, movies are often rented for short or long time periods. Sales may or may not be properly represented quarterly.

PART B:

"Provide original code for function(s) in the text format that perform the transformation(s) you identified in part A4."

CASE WHEN tot_sales >= (SELECT avg(tot_sales) FROM business_extract bri WHERE bre.store_id = bri.store_id)

THEN 'At Or Above Average'

ELSE 'Below Average'

END AS avg_flag

```
INSERT INTO business_parent (store_id, address, category, tot_sales, avg_flag)
SELECT store_id
      ,address
      ,category
      ,tot_sales
      ,CASE WHEN tot_sales >= (SELECT avg(tot_sales) FROM business_extract bri WHERE bre.store_id = bri.store_id)
            THEN 'At Or Above Average'
            ELSE 'Below Average'
            END AS avg_flag
FROM business_extract bre;
```

EXPLANATION:

Part A.4 defines 'avg_flag' as our field with a translation. Here using a correlated subquery if the total sales for each category passes the average total sales per category in each store, then a text field will say such.

PART C:

"Provide original SQL code in a text format that creates the detailed and summary tables to hold your report table sections."

```
CREATE TABLE business_parent(
store_ID INT
  ,address VARCHAR(50)
  ,category CHAR(30)
  ,tot_sales NUMERIC(6,2)
  ,avg_flag CHAR(30)
);

CREATE TABLE business_child(
store_ID INT
  ,category CHAR(30)
  ,tot_sales NUMERIC(6,2)
);
```

EXPLANATION:

The first table, business_parent, is the detailed table. The second table, business_child, is the summary table. No use of primary key for either table.

PART D:

"Provide an original SQL query in a text format that will extract the raw data needed for the detailed section of your report from the source database."

```
CREATE TABLE business_extract(
store_ID INT
  ,tot_sales NUMERIC(6,2)
  ,category CHAR(30)
  ,address VARCHAR(50)
);
```

NOTE: This table is made as an extraction table, so both parent and child tables can be updated upon insertion into the extract table.

```
INSERT INTO business_extract
SELECT sto.store_id
  ,SUM(sto.amount)
  ,sto.name
  ,addr.address
FROM address addr INNER JOIN
(
SELECT staf.staff_id
  ,str.store_id
  ,staf.amount
  ,staf.name
  ,staf.payment_date
  ,staf.address_id
```

```
FROM store str INNER JOIN
(
SELECT stf.staff_id
  ,stf.address_id
  ,stf.store_id
  ,pym.amount
  ,pym.name
  ,pym.payment_date
FROM staff stf INNER JOIN
(
SELECT pay.staff_id -- rented or paid staff, paid?
  ,pay.amount
  ,rnt.name
  ,pay.payment_date
FROM payment pay INNER JOIN
(
SELECT rent.rental_id
  ,rent.rental_date
  ,rent.inventory_id
  ,rent.staff_id
  ,rent.customer_id
  ,invt.name
FROM rental rent INNER JOIN
(
SELECT inv.inventory_id
  ,inv.film_id
  ,inv.store_id
  ,flim.name
```

```
FROM inventory inv INNER JOIN (
SELECT flm.film_id
  ,fc.name
FROM film flm INNER JOIN (
SELECT fct.film_id
  ,fct.category_id
  ,cat.name
FROM film_category fct INNER JOIN (
SELECT category_id
  ,name
FROM category
) cat
ON fct.category_id = cat.category_id
) fc
ON flm.film_id = fc.film_id
) flim
ON inv.film_id = flim.film_id
) invt ON rent.inventory_id = invt.inventory_id
) rnt ON pay.customer_id = rnt.customer_id AND pay.rental_id = rnt.rental_id -- orders paid in 2007 (most current year), 14592 rows returned, same as payment table records.
) pym ON stf.staff_id = pym.staff_id
) staf ON staf.store_id = str.store_id
) sto ON addr.address_id = sto.address_id -- AND date_part('year', sto.payment_date) = 2007
GROUP BY sto.store_id, sto.name, addr.address
```

EXPLANATION:

The new table, business_extract, has one purpose. This table is where the needed data is extracted. After such is done the following parts using a trigger and procedure will refresh both parent and child tables as the extract table gets new data.

PART E:

"Provide original SQL code in a text format that creates a trigger on the detailed table of the report that will continually update the summary table as data is added to the detailed table."

```
CREATE TRIGGER mig_bus_tables
AFTER INSERT OR UPDATE
ON business_extract
FOR EACH ROW
EXECUTE FUNCTION migrate_data ();
```

EXPLANATION:

The trigger 'mig_bus_tables' will go after an insertion or update has been made to the data extract table and execute the function migrate_data. The migrate_data() function will be followed up in part F.

PART F:

"Provide an original stored procedure in a text format that can be used to refresh the data in both the detailed table and summary table. The procedure should clear the contents of the detailed table and summary table and perform the raw data extraction from part D. Identify a relevant job scheduling tool that can be used to automate the stored procedure."

```
CREATE OR REPLACE FUNCTION migrate_data()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
CALL data_update();
RETURN NEW;
END;
$$

CREATE OR REPLACE PROCEDURE data_update()
LANGUAGE plpgsql
AS $$
BEGIN
TRUNCATE TABLE business_parent;
TRUNCATE TABLE business_child;

INSERT INTO business_parent (store_id, address, category, tot_sales, avg_flag)
SELECT store_id
 ,address
 ,category
```

```
 ,tot_sales
 ,CASE WHEN tot_sales >= (SELECT avg(tot_sales) FROM business_extract bri WHERE bre.store_id =
bri.store_id)
THEN 'At Or Above Average'
ELSE 'Below Average'
END AS avg_flag
FROM business_extract bre;

INSERT INTO business_child (store_ID, category, tot_sales)
SELECT store_ID, category, tot_sales FROM business_parent WHERE avg_flag LIKE 'Below%';
END;
$$
```

EXPLANATION:

With the use of the extracted data (table business_extract) we know the trigger on the table calls function ‘migrate_data()’. The migrate_data() function calls a procedure, procedure data_update() that then truncates the detailed and summary table before adding data to fields defined in part A.

INDENTIFYING:

A relevant job scheduling tool involves the PostGreSQL pgAgent, which is found often with pgAdmin 4 but is a separated installation.

PART G:

"Provide a Panopto video recording that includes the presenter and a vocalized demonstration of the functionality of the code used for the analysis."

- https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=a623d2b2-e44b-4761-8eca-aff6016dd0a3

PART H:

"Acknowledge all utilized sources, including any sources of third-party code, using in-text citations and references. If no sources are used, clearly declare that no sources were used to support your submission."

EXPLANATION:

No sources were used to support this submission.