Advanced Data Management - D191

Submission 1 - Derrick Koehn - Western Governors University

A. Summarize one real-world written business report.

For this assessment, I will be demonstrating the extraction, transformation, and loading of data to support a business report. I will create custom SQL code to extract relevant data from tables in a PostgreSQL database, organize that data in a detailed table, and display insights from that detailed table in a smaller summary table. The tables I will create in this assignment will allow management at a video store chain to easily find how many copies of each film are in inventory at each store and how many films in total each store has in inventory. Each month, the company places an order with its supplier to update and replenish its inventory. To do this, management needs to know for each film, how many copies are in stock at each store. They also need to know whether their total inventory for any store has decreased beyond the location's target inventory. Previously, staff at each store location had to manually count how many films were in inventory. This report will eliminate that task and provide management with easier access to the inventory information they need.

A1, A2. Identify and describe the specific fields that will be included in the detailed table and the summary table of the report.

For this assessment, I will create two tables: a detailed table and a summary table.

The detailed table, called film_inventory_by_store, will include the following fields.

- A store_name field. This field will be a varying character field set to contain up to 50 characters. For each entry in the film_inventory_by_store table, this field will contain the name of the store associated with the entry. This name will be created using a user-defined function that will append the word "Blockbuster" to the name of the city in which the store is located. The city name from which the store name will be derived will be extracted from the city table of the dvdrental database via foreign key dependencies linking each store to a city via corresponding entries in the address table.
- A film_name field. This field will be a varying character field set to contain up to 250 characters. For each entry in the film_inventory_by_store table, this field will contain the name of a film associated with the entry. This name will be extracted from the title field of the film table of the dvdrental database.
- An inventory_count field. This field will be a smallint field capable of storing values less than 32,767. For each entry in the film_inventory_by_store table, this field will contain a number showing how many copies of the associated film are in inventory at the associated store. This number will be calculated by counting the number of entries in the inventory table where the film_id field matches the associated film and where the store_id matches the associated store.

The summary table, called store_inventory_summary, will include the following fields.

- A store_name field. This field will be a varying character field set to contain up to 50 characters. For each entry in the store_inventory_summary table, this field will contain the name of the store associated with the entry. This field will be populated

using data from the store_name field of the film_inventory_by_store table described above.
- An inventory_count field. This field will be an integer field capable of storing values less than 2,147,483,647. For each store in the store_inventory_summary table, this field will contain a number showing the total number of films in inventory at that store. This number will be calculated by totaling the inventory_count fields associated with each store in the film_inventory_by_store table.

A3. Identify *at least* two specific tables from the given dataset that will provide the data necessary for the detailed table section and the summary table section of the report.

The data stored in the detailed table will be derived from three tables. The data stored in the store_name field will be extracted from the city table, and transformed by a user-defined function. The data stored in the film_name field will be extracted from the film table. The data stored in the inventory_count field will be created by counting entries, within the inventory table, matching certain criteria.

The data in the summary table will be derived from the detailed table.

A4. Identify *at least* one field in the detailed table section that will require a custom transformation with a user-defined function and explain why it should be transformed (e.g., you might translate a field with a value of *N* to *No* and *Y* to *Yes*).

The data stored in the store_name field of the detailed table will be extracted from the city table, and transformed by a user-defined function that will append the word “BlockBuster” to the end of the city name to create a store name.

A5. Explain the different business uses of the detailed table section and the summary table section of the report.

Each month, management at the video rental company places an order with their film distributor to update and replenish the inventory at each store location. To do this, they need to know how many copies will need to be purchased to meet their inventory targets for each store. A total showing how many films are in inventory at each store will be accessible by viewing the contents of the summary table. Once the quantity that must be ordered has been established, staff will use specific information in the detailed table regarding how many copies of each film are in inventory at each store to decide specifically which films must be ordered, and to which locations they must be delivered. Each of these reports automates what was previously a tedious manual process of counting inventory, saving the company time and money.

A6. Explain how frequently your report should be refreshed to remain relevant to stakeholders.

The report should be refreshed monthly because management uses it to inform their monthly inventory purchasing decisions.

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

The following code creates a user-defined SQL function that appends the word “Blockbuster” to the text it is passed as an argument. For example, calling get_store_name(‘Montreal’) would return ‘Montreal Blockbuster’.

```
CREATE FUNCTION get_store_name(city_name VARCHAR)
RETURNS VARCHAR AS $$
      SELECT city_name || ' Blockbuster';
$$ LANGUAGE sql;
```

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

The detailed table, called film_inventory_by_store is created by executing the following SQL statement.

```
CREATE TABLE film_inventory_by_store (
      store_name VARCHAR(50),
      film_name VARCHAR(250),
      inventory_count SMALLINT
);
```

The summary table, called store_inventory_summary is created by executing the following SQL statement.

```
CREATE TABLE store_inventory_summary (
      store_name VARCHAR(50),
      inventory_count INTEGER
);
```

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.

The following SQL statement extracts needed data from the pre-existing dataset and stores it in the detailed table called film_inventory_by_store.

```
INSERT INTO film_inventory_by_store (store_name, film_name, inventory_count)
SELECT get_store_name(ci.city), f.title, COUNT(*) AS inventory_count
FROM inventory i
JOIN film f ON f.film_id = i.film_id
JOIN store s ON s.store_id = i.store_id
JOIN address ad ON s.address_id = ad.address_id
JOIN city ci ON ad.city_id = ci.city_id
GROUP BY ci.city, f.title
ORDER BY f.title, get_store_name;
```

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.

The following SQL code creates a function called update_store_inventory_summary(). Then, it creates a trigger on the detailed table, film_inventory_by_store, which causes the contents of the summary table, store_inventory_summary, to be updated automatically each time a new entry is added to the detailed table.

```
CREATE FUNCTION update_store_inventory_summary()
RETURNS TRIGGER AS
$$
BEGIN
      DELETE FROM store_inventory_summary;
      INSERT INTO store_inventory_summary (store_name, inventory_count)
      SELECT store_name, SUM(inventory_count)
      FROM film_inventory_by_store
      GROUP BY store_name;
      RETURN NULL;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER update_store_inventory_summary
AFTER INSERT ON film_inventory_by_store
FOR EACH STATEMENT
      EXECUTE FUNCTION update_store_inventory_summary();
```

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.

The following SQL statement creates a stored procedure that clears the contents of both the detailed table and the summary table. Then, it reloads the data into the detailed table using the code from part D (see part D above), which causes the trigger from part E (see Part E above) to refresh the data in the summary table.

```
CREATE PROCEDURE refresh_inventory_tables()
LANGUAGE plpgsql
AS
$$
BEGIN
--Clear both tables--
DELETE FROM film_inventory_by_store;
DELETE FROM store_inventory_summary;
--Reload data into the detailed table. The summary table will be automatically
updated.--
INSERT INTO film_inventory_by_store (store_name, film_name, inventory_count)
SELECT get_store_name(ci.city), f.title, COUNT(*) AS inventory_count
FROM inventory i
JOIN film f ON f.film_id = i.film_id
JOIN store s ON s.store_id = i.store_id
JOIN address ad ON s.address_id = ad.address_id
JOIN city ci ON ad.city_id = ci.city_id
GROUP BY ci.city, f.title
ORDER BY f.title, get_store_name;
END;
$$;
```

F1. Identify a relevant job scheduling tool that can be used to automate the stored procedure.

One commonly used job scheduling tool that works well to schedule tasks for PostgreSQL databases is called pg_cron. Scheduling the automated task created using the SQL code above can be accomplished in several easy steps.

First, you must install the pg_chron extension both on the underlying PostgreSQL database and on the pgAdmin 4 client which is used to interact with the database. Detailed installation instructions can be found at https://github.com/citusdata/pg_cron/blob/main/README.md.

Once the pg_chron extension has been installed, the refresh_inventory_tables() procedure can be automated by executing the following code in pgAdmin 4. The following code is formatted in such a way that the stored procedure would be executed automatically each month. See section A6 of this document for an explanation of why the procedure should run monthly.

```
SELECT cron.schedule('0 0 1 * *', $$ CALL my_stored_procedure(); $$);
```

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

Following is a link to the video presentation.
https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=bbbe009c-d3bf-4de5-ab2c-afda01172840

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.

Throughout this course, I studied a variety of learning materials related to PostgreSQL, databases, and data management. However, I did not quote or paraphrase any sources throughout this document. All the SQL code is original and not copied from any external source. What follows is a list of resources that contributed to my knowledge of the subject matter.

Learning Materials Utilized

Dias, H. (2022, May 4). An overview of job scheduling tools for PostgreSQL. Severalnines. Retrieved April 4, 2023, from https://severalnines.com/database-blog/overview-job-scheduling-tools-postgresql

Malik, U., Goldwasser, M., &amp; Johnston, B. (2019). Sql for data analytics: Perform fast and efficient data analysis with the power of Sql. Packt.

The PostgreSQL Global Development Group. (2023, February 9). PostgreSQL 15.2 documentation. PostgreSQL Documentation. Retrieved April 4, 2023, from https://www.postgresql.org/docs/current/

Slot, M. (n.d.). Pg_chron Documentation. Retrieved April 3, 2023, from https://github.com/citusdata/pg_cron/blob/main/README.md