Jon Hendrix
D191 – Advanced Data Management
Performance Assessment

<u>Section A1</u>

The specific fields that will be utilized in the creation of a detailed table and summary table are as follows: The detailed table will include the rental_id and rental_date fields from the rental table, the title field from the film table, the film_id field from the inventory table, and the name field from the category table renamed as film_category. The summary table will include the newly named film_category field from the category table and a newly created rental_count field that counts the total number of films rented from each film category or genre.

<u>Section A2</u>

The types of data fields for the detailed table are presented are as follows: The rental_id field is an integer datatype and primary key for the rental table that uniquely identifies each row for every instance of a rental. The rental_date field is a timestamp datatype that identifies when the film was rented and is from the rental table. The title field is from the film table and is a varchar datatype that allows up to 255 characters that contains the title of the film. The film_id field is an integer datatype and a primary key that uniquely identifies each film in the film table. The category_id field is from the film_category table and is an integer datatype that uniquely identifies each film category. The name field is from the category table and is a varchar datatype that allows up to 25 characters and describes the category or genre of film. This field has been renamed to film_category.

The types of data fields for the summary table are presented as follows: The name field is being used again from the category table and is a varchar datatype that allows up to 25 characters and describes the category or genre of film. This field has been renamed to film_category. The rental_count field is a newly created field of the integer datatype that counts the total number of films rented from each film category or genre.

Section A3

The film table, category table, rental table, inventory table, and film_category table will be utilized from the DVD rental database to provide the data necessary for both the detailed table and summary table. For the detailed table, the film table is linked to the inventory table through the film_id field and provides the film title. The category table will provide the name field that will be renamed as film_category. The rental table provides rental_id and rental_date indicating the primary key for rentals and when the film was rented. The inventory table functions as a link between the rental and film tables through the film_id field. The film_category table is joined to the inventory table on the film_id field.

For the summary table, the rental table provides the rental_id field which is necessary to count the amount of rentals for the newly created rental_count field. The inventory table functions the same as it does for the detailed table in that it provides the join from the rental table and film table through the film_id field to confirm which films have been rented. The category table is joined to the film_category table through the category_id primary key to get the film category or genre name. Below is a chart outlining the structure of the columns for both the detailed and summary tables.

Source Table	Column Name	Data Type	Destination	Definition
rental	rental_id	INT	detailed_table	Uniquely indentifies each rental
rental	rental_date	TIMESTAMP	detailed_table	Identifies when film was rented
film	title	VARCHAR(255)	detailed_table	Contains title of film
inventory	film_id	INT	detailed_table	Uniquely identifies each film
category	name (renamed film_category)	VARCHAR(25)	detailed_table/summary_table	Describes genre or category of film
summary_table (derived)	rental_count	INT	summary_table	Total count for number of rentals

Section A4

The rental_date field in the detailed table can be transformed from its default timestamp datatype to a varchar datatype to make the date more presentable and readable on a report. The function is named date_fix and uses the TO_CHAR function to easily convert a timestamp to a simple month, day, year format. For example, a timestamp date such as ‘2008-05-26 13:30:48’ would be converted to ‘May 26, 2008’ using this function. When presenting the summary table to management, this will make the table appear more presentable and readable than the default timestamp datatype.

Section A5

There are several business uses of the detailed table and the summary table worth mentioning. The summary table breaks the details down into a simple report and is likely to be presented to management. Knowing which film categories are the most popular and yields the most amount of rentals would prompt store owners to display these categories of films in their main displays in the store, making them easily accessible for customers and thus increasing profits. The summary table could also provide insight about which film genres to promote over others based on how often they are being rented. The detailed table consists of many more data points that may or may not be of interest to management at a given time. A manager could see the summary table but could request to verify or validate the information with more data. Another scenario could involve the manager wanting to see exactly what time of the year films were rented to figure out when best to promote certain film categories. In this situation the detailed table is necessary to either present the information directly to management, or to pull in more data points from the detailed table into the summary.

Section A6

The report should be refreshed on a weekly basis to keep up with new films becoming available as video rentals. This will ensure stakeholders that as new films are being released for home video, they will be on the shelves and available for customers as soon as they are available for the company to purchase and stock on their shelves. Considering the size of the film industry, there would be ample amount of videos becoming available on an almost daily basis, however, refreshing the report on a daily basis wouldn't be realistic as most stores likely operate their inventory deliveries on a weekly or monthly basis. This makes refreshing the report on a weekly basis the most logical choice for the business.

Section B

Below is SQL code for creating a custom transformation described in section A4.

```sql
-- Part B - Custom Function for changing timestamp datatype to varchar
CREATE OR REPLACE FUNCTION date_fix(rental_date timestamp)
RETURNS varchar(25)
LANGUAGE plpgsql
AS $$
BEGIN
RETURN TO_CHAR(rental_date, 'FMMonth DD, YYYY'); --"FM" stands for Fill Mode and
eliminates the extra spaces to better justify the text
END;
$$
-- Part B - Function Call for date_fix, to be used with manual timestamp input or
with an entire column
SELECT date_fix(rental.rental_date)
FROM rental;
```

Section C

Below is SQL code that creates the detailed and summary tables titled detailed_table and summary_table respectively.

```sql
-- Part C - Detailed Table Creation
CREATE TABLE detailed_table (
rental_id INT,
rental_date TIMESTAMP,
title VARCHAR(255),
film_id INT,
film_category VARCHAR(25)
);
-- Part C - Summary Table Creation
CREATE TABLE summary_table (
film_category VARCHAR(25),
rental_count INT
);
```

Section D

Below is a SQL Query that extracts all the data from the database tables and inserts it into the detailed_table and summary_table respectively.

```sql
-- Part D - This will extract the data for the detailed_table and populate it
INSERT INTO detailed_table (rental_id, rental_date, title, film_id,
film_category)
SELECT rental.rental_id, rental.rental_date, film.title, inventory.film_id,
category.name AS film_category
FROM rental
JOIN inventory ON rental.inventory_id = inventory.inventory_id
JOIN film ON inventory.film_id = film.film_id
JOIN film_category ON inventory.film_id = film_category.film_id
JOIN category ON film_category.category_id = category.category_id
ORDER BY rental.rental_id;
-- Part D - This will extract data for the summary_table and populate it
INSERT INTO summary_table (film_category, rental_count)
SELECT category.name AS film_category, COUNT(*) AS rental_count
FROM rental
JOIN inventory ON rental.inventory_id = inventory.inventory_id
JOIN film_category ON inventory.film_id = film_category.film_id
JOIN category ON film_category.category_id = category.category_id
GROUP BY category.name
ORDER BY COUNT(*) DESC;
```

Section E

Below is SQL code that creates a trigger on the detailed_table. This will continually update the summary table as data is added to the detailed table.

```sql
-- Part E - This is the trigger that calls the function for updating the summary table
CREATE TRIGGER summary_table_update_trigger
AFTER INSERT ON detailed_table FOR EACH ROW
EXECUTE FUNCTION summary_table_update();
-- Part E - Function that will be called by the trigger to update detailed_table
CREATE OR REPLACE FUNCTION summary_table_update()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN -- This section recreates the summary table when the trigger is executed
DROP TABLE IF EXISTS summary_table;
CREATE TABLE summary_table AS
SELECT category.name AS film_category, COUNT(*) AS rental_count
FROM rental
JOIN inventory ON rental.inventory_id = inventory.inventory_id
JOIN film_category ON inventory.film_id = film_category.film_id
JOIN category ON film_category.category_id = category.category_id
GROUP BY category.name
ORDER BY COUNT(*) DESC;
END;
$$
```

Section F

Below is a stored procedure titled data_update_refresh() that can be called to refresh the data on both the detailed_table and summary_table.

Section F1

PostgreSQL does not have automation features that can refresh data, so another tool is required to provide automation (Dias, 2020). A tool such as Linux crontab in conjunction with PostgreSQL can achieve this function by automatically running in the background and verifying configuration files that are based on scripts written to execute the automation (Dias, 2020).

```
-- Part F - Stored Procedure for refreshing the data on detailed and summary
tables
CREATE OR REPLACE PROCEDURE data_update_refresh()
LANGUAGE plpgsql
AS $$
BEGIN
-- Clear the contents of both tables
DROP TABLE IF EXISTS detailed_table;
DROP TABLE IF EXISTS summary_table;
-- Recreates detailed_table
CREATE TABLE detailed_table AS
SELECT rental.rental_id, rental.rental_date, film.title, inventory.film_id,
category.name AS film_category
FROM rental
JOIN inventory ON rental.inventory_id = inventory.inventory_id
JOIN film ON inventory.film_id = film.film_id
JOIN film_category ON inventory.film_id = film_category.film_id
JOIN category ON film_category.category_id = category.category_id
ORDER BY rental.rental_id;
-- Recreates summary_table
CREATE TABLE summary_table AS
SELECT category.name AS film_category, COUNT(*) AS rental_count
FROM rental
JOIN inventory ON rental.inventory_id = inventory.inventory_id
JOIN film_category ON inventory.film_id = film_category.film_id
JOIN category ON film_category.category_id = category.category_id
GROUP BY category.name
ORDER BY COUNT(*) DESC;
END;
$$;
-- Part F - Call execution for the stored procedure data_update_refresh()
CALL data_update_refresh;
```

References

Dias, H. (2020, February 3). *An overview of job scheduling tools for PostgreSQL*. severalnines. https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/