Kerrie Abrams

Student ID: 010894830

Task 1

Section A

This report provides insight into the most popular films for each film category, what kinds of films they are, and which of the inventory are rented the most. A detailed view displays the number of rentals for each inventory item which provides insight into which discs are experiencing the most wear. A summarized view displays the most popular films for each genre, as well as the rating information for each of the films. This information will allow the business to make appropriate business decisions for adjusting their inventory based on wear and film popularity.

Section A1-A3

The following table describes the fields found in the detailed view.

Original table	**Data field**	**Data type**
film	film_id	integer
film	title	string
film_category	category name	string
film	rating	mpaa_rating
inventory	inventory_id	integer
rental	Count of rental_id as total_rentals	integer

In addition, the following table describes the data fields found in the summary view.

Original table	**Data field**	**Data type**
film_category	category name	string
film	title	string
film	rating	mpaa_rating
n/a	rating_definition transformed from rating	string
detailed view	Sum of total_rentals as total_ rentals	integer

Section A4

To clarify the meanings of the MPAA film ratings and to better understand the types of films that are the most popular, the field, rating_definition, is included in the summary view. A user-defined function is used on the rating field to transform the MPAA ratings into their respective definitions.

Section A5

The detailed table section of this report can be used to identify which discs need replacement or repair. Discs that experience a greater number of rentals will typically need to be replaced sooner than others. In addition, the information found in this section may be used to identify which films may need additional discs added to the inventory based on how frequently discs are rented out. The summary table section of this report can used to identify which kinds of films are performing the best. This information can used in marketing decisions, such as creating advertisements featuring some of the most popular film types. And just as the detailed table section, it may also be used to identify which films to purchase more copies of or for disc maintenance.

Section A6

This report should be refreshed monthly if the intended use of the results are for inventory maintenance. This will ensure that inventory demands are being met as films become more popular and that any overused discs are being checked for repair or replacement. However, if this report is solely used for marketing purposes, a quarterly refresh may be preferred. A longer time frame with give a more accurate view of what the market generally prefers.

Section B

The following code creates the function explained in Section A4. It is used when populating the summary report to transform an MPAA film rating into a rating definition for clarification. This function uses a simple CASE expression as found in the *PostgreSQL CASE* tutorial (2020).

```
CREATE OR REPLACE FUNCTION rating_definition_function(rating mpaa_rating)
	RETURNS varchar
	LANGUAGE plpgsql
AS
$$
DECLARE rating_definition varchar(45);
BEGIN
	RETURN
	CASE rating
		WHEN 'G' THEN 'General Audiences'
		WHEN 'R' THEN 'Restricted'
		WHEN 'PG' THEN 'Parental Guidance Suggested'
		WHEN 'PG-13' THEN 'Parents Strongly Cautioned'
		WHEN 'NC-17' THEN 'Adults Only'
	END rating_definition;
END;
$$
```

Section C

The following code creates the detailed report table.

```
DROP TABLE IF EXISTS detailed_report;
CREATE TABLE detailed_report(
	film_id int,
	title varchar(225),
	category varchar(25),
	rating mpaa_rating,
	inventory_id int,
	total_rentals int
);
```

The following code creates the summary report table.

```
DROP TABLE IF EXISTS summary_report;
CREATE TABLE summary_report(
	category varchar(25),
	film_id int,
	title varchar(225),
	rating mpaa_rating,
	rating_definition varchar(45),
	total_rentals int
);
```

Section D

The following code extracts data from the database and populates the detailed report table. This statement performs numerous joins on the tables inventory, film, rental, film_category, and category to insert the selected data fields into the detailed_report table. In addition, a COUNT() of rental_id is used to populate the total_rentals column with the total rentals of each disc.

```
INSERT INTO detailed_report
	(film_id,
	title,
	category,
	rating,
	inventory_id,
	total_rentals)
	SELECT f.film_id, f.title, cat.name, f.rating, i.inventory_id, COUNT(r.rental_id)
	FROM inventory i
	LEFT JOIN film f ON f.film_id = i.film_id
	LEFT JOIN rental r ON i.inventory_id = r.inventory_id
	LEFT JOIN film_category fc ON fc.film_id = f.film_id
	LEFT JOIN category cat ON cat.category_id = fc.category_id
	GROUP BY f.film_id, f.title, cat.name, i.inventory_id
	ORDER BY f.film_id
;
```

Section E

The following code creates a trigger and trigger function on the detailed report that updates the summary report when the detailed report is updated. The trigger function clears the contents of the summary report and then populates it from the detailed report.

```
CREATE OR REPLACE FUNCTION update_trigger_function()
        RETURNS TRIGGER
        LANGUAGE plpgsql
AS
$$
BEGIN
        DELETE FROM summary_report;
        INSERT INTO summary_report(
                category,
                film_id,
                title,
                rating,
                rating_definition,
                total_rentals)
        SELECT DISTINCT ON(category)
                category,
                film_id,
                title,
                rating,
                rating_definition_function(rating) as rating_definition,
                SUM(total_rentals) as total_rentals
        FROM detailed_report
        GROUP BY category, film_id, title, rating, rating_definition
        ORDER BY category, total_rentals DESC
        ;
RETURN NEW;
END;
$$

DROP TRIGGER update_summary_trigger ON detailed_report;

CREATE TRIGGER update_summary_trigger
        AFTER INSERT
        ON detailed_report
        FOR EACH STATEMENT
        EXECUTE PROCEDURE update_trigger_function()
;
```

Section F

The following stored procedure refreshes both the detailed and summary report tables. First it clears both tables and then repopulates them to keep information up to date. This report should be refreshed monthly if the intended use of the results are for inventory maintenance or quarterly if used solely for marketing purposes.

```
CREATE OR REPLACE PROCEDURE update_tables_procedure()
	LANGUAGE plpgsql
AS
$$
BEGIN

-- Clears tables

	DELETE FROM detailed_report;
	DELETE FROM summary_report;


-- Populates tables

	INSERT INTO detailed_report
		(film_id,
		title,
		category,
		rating,
		inventory_id,
		total_rentals)
		SELECT f.film_id, f.title, cat.name, f.rating, i.inventory_id, COUNT(r.rental_id)
		FROM inventory i
		LEFT JOIN film f ON f.film_id = i.film_id
		LEFT JOIN rental r ON i.inventory_id = r.inventory_id
		LEFT JOIN film_category fc ON fc.film_id = f.film_id
		LEFT JOIN category cat ON cat.category_id = fc.category_id
		GROUP BY f.film_id, f.title, cat.name, i.inventory_id
		ORDER BY f.film_id
;
END;
$$
```

Section F1

To automatically refresh the two tables, a job scheduling tool, such as pgAgent may be utilized. pgAgent is a tool used to run and manage scheduled jobs which can consist of SQL statements. Once the extension is installed on the database server and configured for use with pgAdmin, a job may be created to automatically run the stored procedure (The pgAdmin Development Team, 2023).

Sources

PostgreSQL CASE. (2020). PostgreSQL Tutorial. https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-case/

The pgAdmin Development Team. (2023). *pgAdmin 4 documentation* (release 7.0). https://ftp.postgresql.org/pub/pgadmin/pgadmin4/v7.0/docs/pgadmin4-7.0.pdf