Emmett Heffernan

D191 - VDM1

Business question: How do the various categories of film each perform in rentals across all locations?

A.

1. The data used for the report will include all rental records, inventory IDs for each film, and the category associated with each film rented.

2. The tables necessary for supplying the data will include the “rental”, “inventory”, “film”, “film\_category”, and “category” tables.

3. The detailed section of the report will include the fields “rental\_id”, category “name”. The summary section of the report will include “COUNT(rental\_id)” for each category “name”.

4. One field in the detailed section that will require a custom transformation is the “name” field. For the summary report to read easier, values will go from “Action” to “Action Films”, “Animation” to “Animated Films”, and so on.

5. The detailed section of the report supplies the ability to see a list of the categories of every individual rental. The summary section of the report supplies an overview of the total rentals per category.

6. This report should be refreshed monthly. Genre specific rental frequency will likely fluctuate at least slightly periodically across the year each month.

B.

— The programming environment used is pgAdmin4, a free open-source relational database management system.

— This SQL code creates the tables to hold the report sections.

CREATE TABLE detailed\_section (

rental\_id INT,

name VARCHAR(2

);

CREATE TABLE summary\_section (

total INT,

category VARCHAR(25)

);

C.

— This SQL query will extract the data necessary for the detailed section for the report.

INSERT INTO detailed\_section (rental\_id, name)

SELECT

rental.rental\_id,

category.name

FROM

rental

LEFT JOIN inventory

ON rental.inventory\_id = inventory.inventory\_id

LEFT JOIN film\_category

ON inventory.film\_id = film\_category.film\_id

LEFT JOIN category

ON film\_category.category\_id = category.category\_id;

— The SQL code supplies erification of the data’s accuracy

SELECT \*

FROM detailed\_section;

D.

SELECT

count(name) AS Rentals,

CASE name

WHEN 'Action' THEN 'Action Films'

WHEN 'Animation' THEN 'Animated Films'

WHEN 'Children' THEN 'Children’s Films'

WHEN 'Classics' THEN 'Classic Films'

WHEN 'Comedy' THEN 'Comedies'

WHEN 'Documentary' THEN 'Documentaries'

WHEN 'Drama' THEN 'Dramas'

WHEN 'Family' THEN 'Family Films'

WHEN 'Foreign' THEN 'Foreign Films'

WHEN 'Games' THEN 'Video Games'

WHEN 'Horror' THEN 'Horror Films'

WHEN 'Music' THEN 'Music Videos'

WHEN 'New' THEN 'New Films'

WHEN 'Sci-Fi' THEN 'Sci-Fi Films'

WHEN 'Sports' THEN 'Sports Films'

WHEN 'Travel' THEN 'Travel Films'

END

FROM detailed\_section

GROUP BY name;

E.

— This SQL code creates a trigger on the detailed table which will update the summary table as per insert on the detailed table. This will include the required transformations found in part A. 4. And created from item D.

CREATE FUNCTION detailed\_section\_trigger\_function()

RETURNS TRIGGER AS $$

BEGIN

DELETE FROM summary\_section;

INSERT INTO summary\_section (

SELECT

count(name) AS Rentals,

CASE name

WHEN 'Action' THEN 'Action Films'

WHEN 'Animation' THEN 'Animated Films'

WHEN 'Children' THEN 'Children’s Films'

WHEN 'Classics' THEN 'Classic Films'

WHEN 'Comedy' THEN 'Comedies'

WHEN 'Documentary' THEN 'Documentaries'

WHEN 'Drama' THEN 'Dramas'

WHEN 'Family' THEN 'Family Films'

WHEN 'Foreign' THEN 'Foreign Films'

WHEN 'Games' THEN 'Video Games'

WHEN 'Horror' THEN 'Horror Films'

WHEN 'Music' THEN 'Music Videos'

WHEN 'New' THEN 'New Films'

WHEN 'Sci-Fi' THEN 'Sci-Fi Films'

WHEN 'Sports' THEN 'Sports Films'

WHEN 'Travel' THEN 'Travel Films'

END

FROM detailed\_section

GROUP BY name);

RETURN NEW;

END;

$$

LANGUAGE PLPGSQL;

CREATE TRIGGER detailed\_section\_trigger

AFTER INSERT

ON detailed\_section

FOR EACH ROW

EXECUTE PROCEDURE detailed\_section\_trigger\_function();

F.

— This SQL code creates a stored procedure that refreshes both sections of the report. Clearing both tables, then performing data extraction, transformation, and loading.

CREATE PROCEDURE report\_update\_procedure()

LANGUAGE PLPGSQL

AS $$

BEGIN

DELETE FROM detailed\_section;

DELETE FROM summary\_section;

INSERT INTO detailed\_section(

rental\_id, name)

SELECT

rental.rental\_id,

category.name

FROM

rental

LEFT JOIN inventory

ON rental.inventory\_id = inventory.inventory\_id

LEFT JOIN film\_category

ON inventory.film\_id = film\_category.film\_id

LEFT JOIN category

ON film\_category.category\_id = category.category\_id;

END;

$$;

1. The stored procedure can be refreshed on a monthly schedule to ensure data freshness and supply indications of how different genres tend to perform over each month. Storing each monthly report could allow further insight into how genres may fluctuate in rental frequency cyclically over various parts of the year. The procedure could be scheduled by creating a SQL event which calls the procedure on the first of every month.

H.

No third-party sources were used.

I.

No additional sources were used.