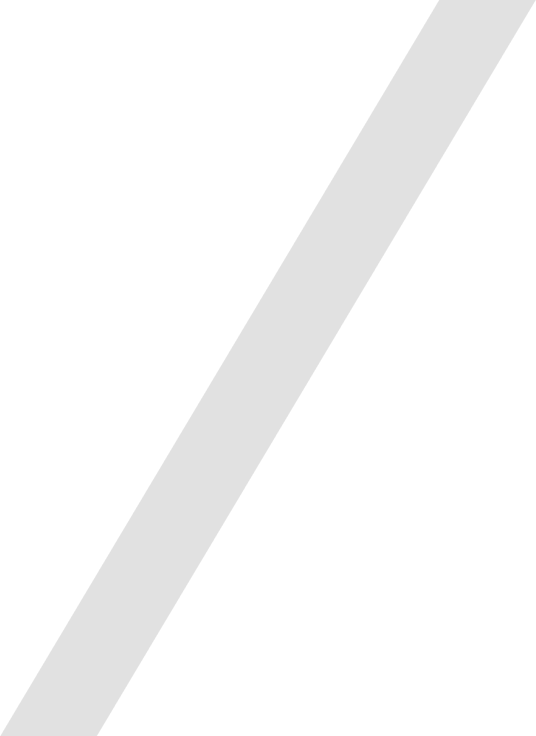
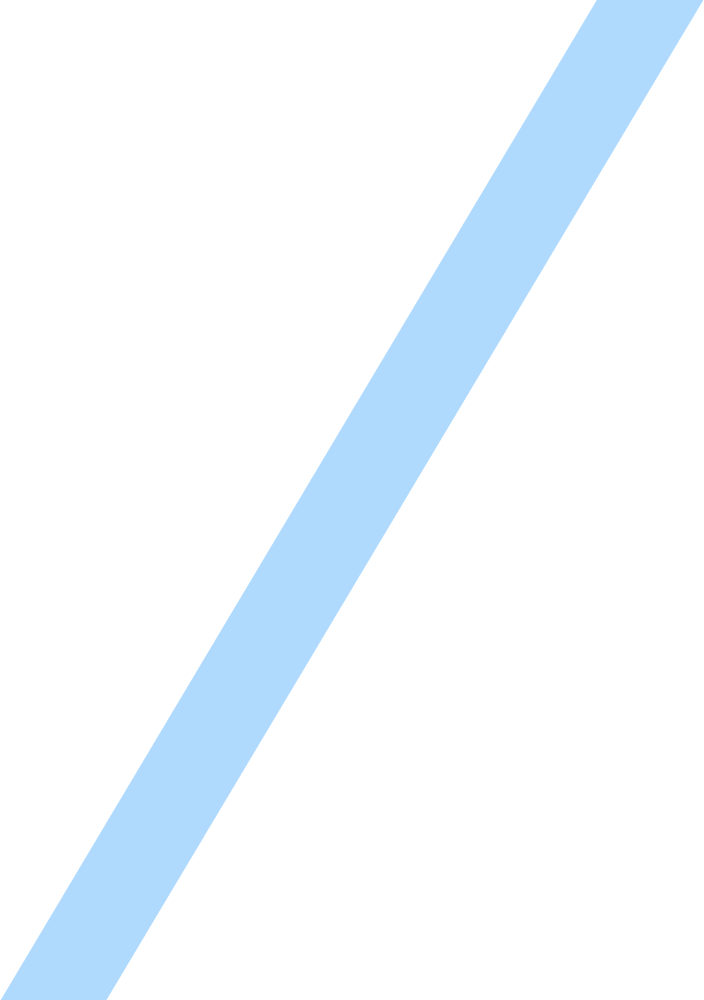
|  |
| --- |
| DVD Business  Report |

|  |
| --- |
| DVDflixBuster  Email: [qtran16@wgu.edu]]  Website: [DVDflixBuster.gg ] |

|  |  |
| --- | --- |
| Quang Tran, Business Report Reporter  Email [qtran16@wgu.edu] |  |



Contents

Contents Page1

Section A2

Section A, B3

Section C, D4

Section E5

Section F6

Cited Works7

|  |
| --- |
| Business Report |

Executive Summary

DVDflixBuster’s top men and women have decided to review the total amount of rentals for every movie in inventory. This summary only applies to movies that have been rented and returned at least one time. In order to identify which movies perform the best overall, these movies will be identified with their rental count along with some additional information about that movie such as MPAA rating, release year, and language.

A.

1. Identify the specific fields that will be included in the detailed table and the summary table of the report.
   1. Detailed table:
      1. *film\_id, title, release\_year, rating, movie\_count*, *movie\_language*
   2. Summary table:
      1. *title, movie\_count\_text,* *rating*
2. Describe the types of data fields used for the report.
   1. Integer, varchar, *mpaa\_rating* (from the database), *bigint*, text
3. 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.
   1. film, inventory, rental
4. 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.
   1. The field *movie\_count* from the detailed table will go through a custom transformation to put the counts into brackets, then labeled as *movie\_count\_text*. When updating the summary table, it should be transformed to make the grouping and reading of movie counts easier to consume. Creating brackets fulfills the purpose of further “summarizing” the counts.
5. Explain the different business uses of the detailed table section and the summary table section of the report.
   1. The detailed table section will provide more in-depth information on the movie and information on which movies have the highest rental count. This can be used to, briefly, see correlations with details such as MPAA rating, language, and whether certain movie-release-years are more prominent in rentals. This information can be used to prioritize, as an example, movies with ratings that are more popular to be rented out, such as NC-17 which are less frequently shown in theaters due to age restrictions. The summary section is a briefer version of the detailed table, highlighting the main goal of the report to identify movie titles, their rating, and rental count in an easier-to-digest tiering system than just raw numbers alone.
6. Explain how frequently your report should be refreshed to remain relevant to stakeholders.
   1. The report can be refreshed daily to account for the most recent daily change in rental numbers, though a monthly cadence can suffice due to the general release schedule of movies.

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

CREATE FUNCTION update\_summary()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

DELETE FROM summary;

INSERT INTO summary (

    SELECT title,

--transforms movie\_count from detailed into movie\_count\_text for summary, putting the values into brackets that can be more easily defined and grouped later if desired

    CASE

        WHEN movie\_count >= 0 AND movie\_count < 10

            THEN '0 to 9'

        WHEN movie\_count >= 10 AND movie\_count < 20

            THEN '10 to 19'

        WHEN movie\_count >= 20 AND movie\_count < 30

            THEN '20 to 29'

        WHEN movie\_count >= 30

            THEN 'Over 30'

    END AS movie\_count\_text,

    rating

    FROM detailed

    GROUP BY movie\_count, title, rating

    ORDER BY movie\_count DESC

);

RETURN NEW;

END; $$

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

--DETAILED SECTION (citation 4)

CREATE TABLE detailed (

    film\_id integer,

    title varchar(255),

    release\_year integer,

    rating mpaa\_rating,

    movie\_count bigint,

    movie\_language text

);

--SUMMARY SECTION

CREATE TABLE summary (

    title varchar(255),

    movie\_count\_text text,

    rating mpaa\_rating

);

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.

    SELECT f.film\_id, f.title, f.release\_year, f.rating,

COUNT(f.title) AS movie\_count,

        CASE

            WHEN f.language\_id = '1'

            THEN 'English'

            ELSE 'we do not rent non-english movies'

        END AS movie\_language

    FROM rental AS r

    INNER JOIN inventory AS i

        ON i.inventory\_id = r.inventory\_id

    INNER JOIN public.film AS f

        ON f.film\_id = i.film\_id

    WHERE r.return\_date IS NOT NULL

    GROUP BY f.title, f.rating, f.release\_year, f.film\_id

    ORDER BY movie\_count DESC

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. (citations 2, 3, 5)

CREATE TRIGGER update\_detailed

    AFTER INSERT ON detailed

    FOR EACH STATEMENT

    EXECUTE PROCEDURE update\_summary();

CREATE FUNCTION update\_summary()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

DELETE FROM summary;

INSERT INTO summary (

    SELECT title,

    CASE

        WHEN movie\_count >= 0 AND movie\_count < 10

            THEN '0 to 9'

        WHEN movie\_count >= 10 AND movie\_count < 20

            THEN '10 to 19'

        WHEN movie\_count >= 20 AND movie\_count < 30

            THEN '20 to 29'

        WHEN movie\_count >= 30

            THEN 'Over 30'

    END AS movie\_count\_text,

    rating

    FROM detailed

    GROUP BY movie\_count, title, rating

    ORDER BY movie\_count DESC

);

RETURN NEW;

END; $$

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.

CREATE PROCEDURE refresh\_tables ()

    LANGUAGE plpgsql AS $$

    BEGIN

    DELETE FROM detailed;

    INSERT INTO detailed (

    SELECT f.film\_id, f.title, f.release\_year, f.rating, COUNT(f.title) AS movie\_count,

        CASE

            WHEN f.language\_id = '1'

            THEN 'English'

            ELSE 'we do not rent non-english movies'

        END AS movie\_language

    FROM rental AS r

    INNER JOIN inventory AS i

        ON i.inventory\_id = r.inventory\_id

    INNER JOIN public.film AS f

        ON f.film\_id = i.film\_id

    WHERE r.return\_date IS NOT NULL

    GROUP BY f.title, f.rating, f.release\_year, f.film\_id

    ORDER BY movie\_count DESC

    );

    END; $$

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

* 1. Because PostgreSQL does not have a built-in scheduler, we must use an added tool. There are options such as: Linux crontab, Agent pgAgent, Extension pg\_cron, or Timescale’s implementation of a scheduler. (citations 6,7)

CITED SOURCES:

1. “PostgreSQL Create Procedure.” PostgreSQL Tutorial, https://www.postgresqltutorial.com/postgresql-plpgsql/postgresql-create-procedure/.
2. “PostgreSQL Create Trigger.” PostgreSQL Tutorial, https://www.postgresqltutorial.com/postgresql-triggers/creating-first-trigger-postgresql/.
3. “PostgreSQL Triggers.” PostgreSQL Tutorial, https://www.postgresqltutorial.com/postgresql-triggers/.
4. “The SQL CREATE TABLE Statement.” SQL Create Table Statement, https://www.w3schools.com/sql/sql\_create\_table.asp.
5. “The SQL INSERT INTO Statement.” SQL Insert into Statement, <https://www.w3schools.com/sql/sql_insert.asp>.
6. “An Overview of Job Scheduling Tools for PostgreSQL” <https://severalnines.com/blog/overview-job-scheduling-tools-postgresql/>
7. “The PostgreSQL Job Scheduler You Always Wanted (But Be Careful What You Ask For)” https://www.timescale.com/blog/the-postgresql-job-scheduler-you-always-wanted-but-be-careful-what-you-ask-for/