# Section A

**Summarize one real-world written business report that can be created from the DVD Dataset from the “Labs on Demand Assessment Environment and DVD Database” attachment.**

*Analysis of Top-Performing Movies by Genre:* The report identifies the top ten highest-grossing movies in each genre, providing a clear picture of which movies are currently most profitable. The strategic insights that can be drawn from their performance can be used to guide future content acquisition and promotional activities. The genre-wise breakdown allows for targeted analysis, enabling a focused strategy for each category of content in the increasingly fragmented media consumption landscape.

**1. Identify the specific fields that will be included in the detailed table and the summary table of the report.**

**Detailed Table (detailed\_movie\_revenue)**

* **film\_id (INT)**
* **title (VARCHAR)**
* **genre (VARCHAR) category.name**
* **total\_revenue (TEXT)**

**Summary Table (summary\_genre\_revenue)**

* **genre (VARCHAR) category.name**
* **top\_ten\_movies (TEXT)**

**2. Describe the types of data fields used for the report.**

* **INT – Integer:**

**Field definition: Represents a whole number with no decimal point.**

**Used in columns that store numerical values that are whole numbers. Such as “film\_id”**

* **VARCHAR – Variable length string:**

**Filed definition: A String that stores up to 255 characters, numbers, or symbols.**

**Used in columns that store relatively short strings of text. Such as “title”, “genre”.**

* **TEXT – Variable length string:**

**Field definition: A string with no predefined length that can store long-format text.**

**Used in columns that store concatenated fields that may vary significantly in length. Such as “total\_revenue”, “top\_ten\_movies”.**

* **NUMERIC – Precise numerical value:**

**Filed definition: Represents a large number including significant digits and decimals.**

**Used in columns where numerical precision is important. Such as my “format\_dollar” function.**

**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.**

**Detailed Table:** Data is derived from the “film”, “film\_category”, “category”, “inventory”, “rental”, and “payment” tables.

**Summary Table:** Data is derived from the detailed\_movie\_revenue table.

**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 (e.g., you might translate a field with a value of N to No and Y to Yes).**

The “total\_revenue” field in the “detailed\_movie\_revenue” table requires a custom transformation. The “format\_dollar” function is used to convert numerical revenue data into a formatted string with a dollar sign for readability to avoid any confusion.

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

**Detailed Table:** Provides in-depth information on individual film performance, including revenue. Useful for analyzing which specific films are most successful financially within each genre.

**Summary Table:** Offers a quick, high-level view of the top-performing films in each genre. Useful for strategic decisions about which genres are most lucrative and merit further investment or marketing focus.

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

The report should be refreshed on a monthly basis. Frequent updates are essential to capture the latest trends, especially in a fast-changing market like movie rentals, and to enable timely decision making based on the most current data. A monthly refresh gives us the latest popularity trends and revenue info without overworking the staff or bogging down the system.

# Section B

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

-- Section B: Function for Formatting Dollar Amounts

CREATE OR REPLACE FUNCTION format\_dollar(amount NUMERIC)

RETURNS TEXT AS $$

BEGIN

-- Formats numeric amount to a dollar string format. "FM" Fill Mode used to eliminate white space.

RETURN '$' || TO\_CHAR(amount, 'FM999,999,999.00');

END;

$$ LANGUAGE plpgsql;

-- Test the format\_dollar function

SELECT format\_dollar(123456.78);

# Section C

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

-- Section C: Table Creation

DROP TABLE IF EXISTS detailed\_movie\_revenue;

CREATE TABLE detailed\_movie\_revenue (

film\_id INT PRIMARY KEY,

title VARCHAR(255),

genre VARCHAR(255),

total\_revenue TEXT

);

DROP TABLE IF EXISTS summary\_genre\_revenue;

CREATE TABLE summary\_genre\_revenue (

genre VARCHAR(255) PRIMARY KEY,

top\_ten\_movies TEXT

);

-- Verify tables are created.

SELECT \* FROM detailed\_movie\_revenue;

SELECT \* FROM summary\_genre\_revenue;

# Section 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.**

-- Section D: Data Insertion with Ranking

-- Clear existing data from detailed\_movie\_revenue for a fresh start.

TRUNCATE TABLE detailed\_movie\_revenue;

-- Rank and insert top 10 movies by revenue in each genre

WITH RankedMovies AS (

SELECT

f.film\_id,

f.title,

c.name AS genre,

format\_dollar(SUM(p.amount)) AS total\_revenue,

ROW\_NUMBER() OVER (PARTITION BY c.name ORDER BY SUM(p.amount) DESC) as rank

FROM film f

JOIN film\_category fc ON f.film\_id = fc.film\_id

JOIN category c ON fc.category\_id = c.category\_id

JOIN inventory i ON f.film\_id = i.film\_id

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

JOIN payment p ON r.rental\_id = p.rental\_id

GROUP BY f.film\_id, f.title, c.name

)

INSERT INTO detailed\_movie\_revenue (film\_id, title, genre, total\_revenue)

SELECT film\_id, title, genre, total\_revenue

FROM RankedMovies

WHERE rank <= 10;

-- Verification

# SELECT \* FROM detailed\_movie\_revenue LIMIT 160;

# Section 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.**

-- Section E: Trigger Creation

CREATE OR REPLACE FUNCTION update\_summary\_table()

RETURNS TRIGGER AS $$

BEGIN

DELETE FROM summary\_genre\_revenue;

INSERT INTO summary\_genre\_revenue (genre, top\_ten\_movies)

SELECT

genre,

STRING\_AGG(title, ', ') AS top\_ten\_movies

FROM detailed\_movie\_revenue

GROUP BY genre;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Defines the trigger to call the above function

CREATE TRIGGER trigger\_update\_summary

AFTER INSERT OR UPDATE OR DELETE ON detailed\_movie\_revenue

FOR EACH ROW EXECUTE FUNCTION update\_summary\_table();

-- Verification

INSERT INTO detailed\_movie\_revenue (film\_id, title, genre, total\_revenue) VALUES (999, 'Test Movie', 'Test Genre', '$1000.00');

SELECT \* FROM summary\_genre\_revenue;

# Section 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.**

-- Section F

CREATE OR REPLACE PROCEDURE refresh\_movie\_data()

LANGUAGE plpgsql AS $$

BEGIN

-- Clear the contents of the detailed and summary tables

TRUNCATE TABLE detailed\_movie\_revenue;

TRUNCATE TABLE summary\_genre\_revenue;

-- Insert the top 10 ranked movies from each genre

INSERT INTO detailed\_movie\_revenue (film\_id, title, genre, total\_revenue)

SELECT

film\_id,

title,

genre,

total\_revenue

FROM (

SELECT

f.film\_id,

f.title,

c.name AS genre,

format\_dollar(SUM(p.amount)) AS total\_revenue,

ROW\_NUMBER() OVER (PARTITION BY c.name ORDER BY SUM(p.amount) DESC) as rank

FROM film f

JOIN film\_category fc ON f.film\_id = fc.film\_id

JOIN category c ON fc.category\_id = c.category\_id

JOIN inventory i ON f.film\_id = i.film\_id

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

JOIN payment p ON r.rental\_id = p.rental\_id

GROUP BY f.film\_id, f.title, c.name

) AS RankedMovies

WHERE rank <= 10;

DELETE FROM summary\_genre\_revenue;

INSERT INTO summary\_genre\_revenue (genre, top\_ten\_movies)

SELECT

genre,

STRING\_AGG(title, ', ') AS top\_ten\_movies

FROM detailed\_movie\_revenue

GROUP BY genre;

END;

$$;

-- Final verification

CALL refresh\_movie\_data();

SELECT \* FROM detailed\_movie\_revenue LIMIT 161;

SELECT \* FROM summary\_genre\_revenue LIMIT 17;

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

The pgAgent is a job scheduling tool for PostgreSQL. it is used to execute stored procedures, SQL statements, and shell scripts. It runs as a daemon on Linux systems, checking for scheduled jobs in the database, but requires manual installation as it's not included with pgAdmin by default. New movies are released all the time, but it takes at least a few weeks for trends to appear. To that end, updating the data once a month would be ideal for analyzing trends at an appropriate interval.

# Section G

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

Panopto link:

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=0d5bdc90-b315-43da-9227-b0fb0015891d

# Section 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.**

PostgreSQL. (n.d.). Retrieved January 2024, from <https://www.postgresql.org>

This is the only site that I used outside of the included course materials. It is an excellent reference site but I did not directly take any code from there.