## **Section A: Business Report Summary**

### **Project Overview:**

This project focuses on answering the business question: **"What types of films are offered?"** The goal is to provide the DVD rental business with insights into its film catalog, specifically focusing on genres and related metrics. By understanding the types of films offered, the business can make informed decisions on inventory management, promotional activities, and pricing strategies. The detailed report will provide a granular view of each film, while the summary report will aggregate data to offer high-level insights. This information will help the business optimize its offerings, attract more customers, and improve overall revenue by ensuring the right genres and films are available to meet customer demand.

### **Business Benefit:**

The report will help the DVD rental business:

* Analyze the distribution of genres to ensure the inventory matches customer preferences.
* Identify popular genres for targeted promotions or pricing adjustments.
* Ensure underrepresented genres are stocked appropriately to meet customer demand.
* Help the business understand which genres have the highest rental rates and durations to make data-driven inventory decisions.

### **Business Question:**

**What types of films are offered?**

### **A1: Specific Fields for the Detailed and Summary Tables**

#### **Detailed Table Fields:**

1. **film\_id** (Integer, Primary Key): A unique identifier for each film.
2. **title** (Variable-length String): The title of the film.
3. **description** (Text): A brief description of the film.
4. **release\_year** (Integer): The year the film was released.
5. **language** (Variable-length String): The language in which the film is available.
6. **rental\_duration** (Integer): The duration for which the film can be rented, in days.
7. **rental\_rate** (Numeric, 5,2): The cost to rent the film.
8. **length** (Integer): The length of the film in minutes.
9. **rating** (Text): The film’s rating (e.g., G, PG, R).
10. **special\_features** (Text): Any special features included with the film.
11. **category** (Variable-length String): The genre of the film.

#### **Summary Table Fields:**

1. **genre** (Variable-length String, Primary Key): The genre of the films.
2. **total\_films** (Integer): The total number of films in each genre.
3. **average\_rental\_rate** (Numeric, 5,2): The average rental rate for films in each genre.
4. **average\_rental\_duration** (Numeric, 5,2): The average rental duration for films in each genre.

### **A2: Types of Data Fields**

* **film\_id**: Integer, unique identifier for each film.
* **title**: Variable-length string (255 characters), the name of the film.
* **description**: Text, a brief description of the film.
* **release\_year**: Integer, the year the film was released.
* **language**: Variable-length string (50 characters), the language in which the film is available.
* **rental\_duration**: Integer, the number of days the film can be rented.
* **rental\_rate**: Numeric (5, 2), the cost to rent the film.
* **length**: Integer, the length of the film in minutes.
* **rating**: Text, the rating of the film (e.g., G, PG, R).
* **special\_features**: Text, any special features associated with the film (e.g., Deleted Scenes, Commentary).
* **category**: Variable-length string (50 characters), the genre of the film.
* **total\_films**: Integer, the total number of films in each genre (summary table).
* **average\_rental\_rate**: Numeric (5, 2), the average rental rate for each genre (summary table).
* **average\_rental\_duration**: Numeric (5, 2), the average rental duration for each genre (summary table).

### **A3: Source Tables for the Detailed and Summary Tables**

1. **film table**: Provides film-specific data such as film\_id, title, description, release\_year, rental\_duration, rental\_rate, length, rating, and special\_features.
2. **film\_category table**: Links films to their genres through film\_id and category\_id.
3. **category table**: Provides the genre name, stored in the category field.
4. **language table**: Provides the name of the language through language\_id.

* **Detailed Table**: All fields in the detailed table are sourced from the film, film\_category, category, and language tables.
* **Summary Table**: The fields in the summary table are aggregated from the detailed\_films table (e.g., total\_films, average\_rental\_rate, average\_rental\_duration).

### **A4: Field Transformation**

* **Field Name**: rating
* **Source Table**: The rating field originates from the film table and is transformed when inserted into the detailed\_films table.
* **Transformation**: The rating field will be transformed into more descriptive text (e.g., "PG-13" becomes "Parents Strongly Cautioned").
* **Reason**: The transformation provides a clearer, more user-friendly description of the rating system, making the data more accessible for non-technical stakeholders.

### **A5: Business Uses of the Detailed and Summary Tables**

1. **Detailed Table**:
   * **Use Case**: The detailed table will be used for in-depth analysis of each individual film. It provides comprehensive information that can help managers understand trends in specific film attributes, such as rental rates, ratings, and languages. This information could be used to make decisions on pricing adjustments or to stock more films in high-demand languages or genres.
2. **Summary Table**:
   * **Use Case**: The summary table offers a high-level view of the film catalog, allowing managers to quickly assess the distribution of genres, the average rental rates, and rental durations for each genre. Stakeholders can use this information to identify which genres are performing well and adjust promotional efforts accordingly. For instance, the company could run genre-specific marketing campaigns or adjust pricing for underperforming genres.

### **A6: Report Refresh Frequency**

* **Recommended Frequency**: The report should be refreshed **weekly**. This frequency ensures that the data remains up-to-date and relevant, while aligning with the business use cases of managing inventory and adjusting pricing or promotions. Since rental data and film inventory do not fluctuate daily, a weekly refresh provides the right balance between data relevance and performance.

### **Part B: Provide Original Code for the User-Defined Function**

The function is designed to transform the rating field in the film table to a more user-friendly format. This transformation is used when populating the detailed\_films table.

#### **B: User-Defined Function for Transforming Ratings**

CREATE OR REPLACE FUNCTION transform\_rating(input\_rating TEXT)

RETURNS TEXT AS $$

BEGIN

RETURN CASE input\_rating

WHEN 'G' THEN 'General Audience'

WHEN 'PG' THEN 'Parental Guidance Suggested'

WHEN 'PG-13' THEN 'Parents Strongly Cautioned'

WHEN 'R' THEN 'Restricted'

WHEN 'NC-17' THEN 'Adults Only'

ELSE 'Unrated'

END;

END;

$$ LANGUAGE plpgsql;

#### **Explanation:**

* This function takes the rating field (e.g., PG, R) from the source data and transforms it into a more descriptive label (e.g., PG becomes "Parental Guidance Suggested").
* **Tested**: This function was tested and works correctly when called during data extraction.
* **Usage**: The transformed rating will be used in the detailed\_films table, making the data more accessible to business users.

### **Part C: Provide SQL Code to Create the Detailed and Summary Tables**

The detailed and summary tables are created based on the fields defined in **Part A**. The detailed table contains granular data for each film, and the summary table aggregates this data by genre.

#### **C: SQL Code to Create the Detailed Table**

DROP TABLE IF EXISTS detailed\_films;

CREATE TABLE detailed\_films (

film\_id SERIAL PRIMARY KEY,

title VARCHAR(255) NOT NULL,

description TEXT,

release\_year INTEGER,

language VARCHAR(50),

rental\_duration INTEGER,

rental\_rate NUMERIC(5, 2),

length INTEGER,

rating VARCHAR(50),

special\_features TEXT,

category VARCHAR(50)

);

**C: SQL Code to Create the Summary Table**

DROP TABLE IF EXISTS summary\_films;

CREATE TABLE summary\_films (

genre VARCHAR(50) PRIMARY KEY,

total\_films INTEGER,

average\_rental\_rate NUMERIC(5, 2),

average\_rental\_duration NUMERIC(5, 2)

);

#### **Explanation:**

#### **Detailed Table:** Contains all film data without aggregation, with one row per film.

#### **Summary Table:** Aggregates the data by genre, including total films, average rental rate, and average rental duration.

#### 

### **Part D: SQL Query to Extract Raw Data for the Detailed Section**

#### This query extracts data from the film, language, film\_category, and category tables, transforming the rating field using the transform\_rating function.

#### **D: SQL Query to Populate the Detailed Table**

INSERT INTO detailed\_films (title, description, release\_year, language, rental\_duration, rental\_rate, length, rating, special\_features, category)

SELECT

f.title,

f.description,

f.release\_year,

l.name AS language,

f.rental\_duration,

f.rental\_rate,

f.length,

transform\_rating(f.rating:: TEXT) AS rating,

f.special\_features,

c.name AS category

FROM

film f

JOIN

language l ON f.language\_id = l.language\_id

JOIN

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

JOIN

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

#### **Explanation:**

* This query pulls data from the film, film\_category, category, and language tables, ensuring at least two tables are used (as required).
* The rating field is transformed using the transform\_rating function, ensuring consistency with Part B.
* The data is inserted into the detailed\_films table, as specified in Part C.

### **Part E: SQL Code to Create a Trigger on the Detailed Table**

The trigger ensures that whenever data is added, updated, or deleted in the detailed\_films table, the summary\_films table is updated accordingly.

#### **E: SQL Code to Create the Trigger and Trigger Function**

##### **1. Create the Trigger Function**

CREATE OR REPLACE FUNCTION update\_summary\_table()

RETURNS TRIGGER AS $$

BEGIN

-- Delete the current summary data for the genre being affected

DELETE FROM summary\_films WHERE genre = NEW.category;

-- Recalculate and insert the updated summary data for the genre

INSERT INTO summary\_films (genre, total\_films, average\_rental\_rate, average\_rental\_duration)

SELECT

category AS genre,

COUNT(\*) AS total\_films,

AVG(rental\_rate) AS average\_rental\_rate,

AVG(rental\_duration) AS average\_rental\_duration

FROM

detailed\_films

WHERE

category = NEW.category

GROUP BY

category;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

**2. Create the Trigger**

CREATE TRIGGER trg\_update\_summary

AFTER INSERT OR UPDATE OR DELETE ON detailed\_films

FOR EACH ROW

EXECUTE FUNCTION update\_summary\_table();

#### **Explanation:**

* **Trigger Function:** Updates the summary\_films table by recalculating the totals and averages for the affected genre whenever a row is inserted, updated, or deleted in the detailed\_films table.
* **Trigger:** Ensures that the function is executed after any modification to the detailed\_films table.

### **Part F: Stored Procedure to Refresh Both Tables**

The stored procedure clears both the detailed\_films and summary\_films tables and re-populates them with fresh data from the source tables.

#### **F: Stored Procedure to Refresh Data**

CREATE OR REPLACE PROCEDURE refresh\_film\_data()

LANGUAGE plpgsql

AS $$

BEGIN

-- Clear existing

TRUNCATE TABLE detailed\_films, summary\_films;

-- Re-populate the detailed\_films table with fresh data

INSERT INTO detailed\_films (title, description, release\_year, language, rental\_duration, rental\_rate, length, rating, special\_features, category)

SELECT

f.title,

f.description,

f.release\_year,

l.name AS language,

f.rental\_duration,

f.rental\_rate,

f.length,

transform\_rating(f.rating:: TEXT) AS rating,

f.special\_features,

c.name AS category

FROM

film f

JOIN

language l ON f.language\_id = l.language\_id

JOIN

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

JOIN

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

-- Step 3: Insert new genres into summary\_films

INSERT INTO summary\_films (genre, total\_films, average\_rental\_rate, average\_rental\_duration)

SELECT

category AS genre,

COUNT(film\_id) AS total\_films,

AVG(rental\_rate) AS average\_rental\_rate,

AVG(rental\_duration) AS average\_rental\_duration

FROM

detailed\_films

GROUP BY

category

-- Only insert rows for new genres (not existing ones)

ON CONFLICT DO NOTHING;

-- Step 4: Update existing genres in summary\_films

UPDATE summary\_films sf

SET

total\_films = df.total\_films,

average\_rental\_rate = df.average\_rental\_rate,

average\_rental\_duration = df.average\_rental\_duration

FROM (

SELECT

category AS genre,

COUNT(film\_id) AS total\_films,

AVG(rental\_rate) AS average\_rental\_rate,

AVG(rental\_duration) AS average\_rental\_duration

FROM

detailed\_films

GROUP BY

category

) AS df

WHERE sf.genre = df.genre;

END;

$$;

#### **Explanation:**

* The procedure **clears both tables** using TRUNCATE and then re-populates them with fresh data.
* The **detailed\_films** table is repopulated from the source tables.
* The **summary\_films** table is populated based on an aggregation of the data in the detailed\_films table.

### **Part F1: Job Scheduling Tool**

### I would use pg\_cron as the job scheduling tool to automate the refresh\_film\_data stored procedure. pg\_cron is a PostgreSQL extension that allows scheduling and running SQL commands at regular intervals directly from within the PostgreSQL environment.

### This tool integrates seamlessly with PostgreSQL and can be used to automate tasks such as refreshing the data weekly without manual intervention. The procedure could be scheduled to run every Monday at 3:00 AM using the following command:

SELECT cron.schedule('weekly\_refresh', '0 3 \* \* 1', $$CALL refresh\_film\_data();$$);

This schedules the procedure to run every Monday at 3:00 AM.

### **Part H: Sources**

#### **Indication of Sources Used:**

* **No outside sources were used** in the development of this project. All the code and content were based on my original work and knowledge of SQL and PostgreSQL, combined with the course material provided.