D326 Task-1 Advanced Data Management

Trayvonious Pendleton

ID 011205284

WGU Email: [tpend32@wgu.edu](mailto:tpend32@wgu.edu)

Part A

For this task, I will analyze the revenue generated by different film categories over a period of time. The business question we are answering is:

"Which film categories generate the most revenue on a monthly basis, and how do they perform over time?"

This report will help DVD rental business stakeholders understand which categories are the most profitable and assist in inventory management, marketing, and promotion focus decisions.

Part A1

**Detailed Table**: This table will include detailed data on each rental transaction.

* rental\_id (Primary key for each rental)
* rental\_date (Date when the DVD was rented)
* film\_id (The film being rented)
* category\_name (The category of the film)
* customer\_id (Customer who rented the DVD)
* payment\_amount (Amount paid by the customer)

**Summary Table**: This table will contain monthly aggregated data on revenue per category.

* month (Month of rental)
* category\_name (The film category)
* total\_rentals (Total number of rentals per category per month)
* total\_revenue (Sum of rental payments per category per month)

Part A2

**Types of Data Fields:**

* **rental\_id**: Integer (Unique identifier for each rental)
* **rental\_date**: Date (The date when a DVD was rented)
* **film\_id**: Integer (Unique identifier for each film)
* **category\_name**: Text (Name of the film category)
* **customer\_id**: Integer (Unique identifier for each customer)
* **payment\_amount**: Decimal (The amount paid by the customer)
* **month**: Date (Truncated to the first day of each month for aggregation purposes)
* **total\_rentals**: Integer (Count of total rentals)
* **total\_revenue**: Decimal (Sum of payments for each category in the month)

Part A3

**Tables Providing Data:**

* **rental table:** Contains rental information, including rental\_id, rental\_date, customer\_id, and inventory\_id.
* **film\_category table**: Connects films to their respective categories, linking film\_id to category\_id.
* **Category table:** Contains the names of the categories**.**
* **Payment table:** Contains payment information, including rental\_id and amount paid.

Part A4

A custom transformation will be required for the month field in the summary table. We'll need to extract the month from the rental\_date in the detailed table to aggregate the data by month. This transformation is necessary because the source data contains the full rental date, but the report requires aggregated monthly summaries.

Part A5

**Business Uses:**

**Detailed Table**: Useful for operational teams who need to track individual rentals, customer activity, and revenue per rental. This helps in understanding specific transactions and identifying customer trends.

**Summary Table**: Useful for upper management, providing high-level insights into which film categories are the most profitable. This can guide decisions on which categories to promote, expand, or reduce in stock.

Part A6

**Refresh Frequency:**

**Monthly**: The report should be refreshed at the end of each month to provide the latest insights into the performance of film categories over the previous month. A monthly refresh aligns with the monthly aggregation of the summary table.

Part B

CREATE OR REPLACE FUNCTION extract\_month(rental\_date DATE) RETURNS DATE AS $$

BEGIN

RETURN DATE\_TRUNC('month', rental\_date);

END; $$ LANGUAGE plpgsql;

Part C

-- Create a detailed rental report table

CREATE TABLE detailed\_rental\_report (

rental\_id INT PRIMARY KEY,

rental\_date DATE,

film\_id INT,

category\_name TEXT,

customer\_id INT,

payment\_amount DECIMAL

);

-- Create the summary report table

CREATE TABLE summary\_rental\_report (

month DATE,

category\_name TEXT,

total\_rentals INT,

total\_revenue DECIMAL

);

Part D

-- Populate the detailed table with rental data

INSERT INTO detailed\_rental\_report (rental\_id, rental\_date, film\_id, category\_name, customer\_id, payment\_amount)

SELECT

r.rental\_id,

r.rental\_date,

f.film\_id,

c.name AS category\_name,

r.customer\_id,

p.amount AS payment\_amount

FROM

rental r

JOIN

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

JOIN

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

JOIN

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

JOIN

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

JOIN

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

Part E

CREATE OR REPLACE FUNCTION update\_summary\_table() RETURNS TRIGGER AS $$

BEGIN

INSERT INTO summary\_rental\_report (month, category\_name, total\_rentals, total\_revenue)

VALUES (

extract\_month(NEW.rental\_date),

NEW.category\_name,

(SELECT COUNT(\*) FROM detailed\_rental\_report WHERE extract\_month(rental\_date) = extract\_month(NEW.rental\_date) AND category\_name = NEW.category\_name),

(SELECT SUM(payment\_amount) FROM detailed\_rental\_report WHERE extract\_month(rental\_date) = extract\_month(NEW.rental\_date) AND category\_name = NEW.category\_name)

)

ON CONFLICT (month, category\_name) DO UPDATE

SET

total\_rentals = EXCLUDED.total\_rentals,

total\_revenue = EXCLUDED.total\_revenue;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER rental\_summary\_update

AFTER INSERT ON detailed\_rental\_report

FOR EACH ROW

EXECUTE FUNCTION update\_summary\_table();

Part F

CREATE OR REPLACE PROCEDURE refresh\_rental\_report() AS $$

BEGIN

-- Clear the detailed table

TRUNCATE TABLE detailed\_rental\_report;

-- Repopulate the detailed table

INSERT INTO detailed\_rental\_report (rental\_id, rental\_date, film\_id, category\_name, customer\_id, payment\_amount)

SELECT

r.rental\_id,

r.rental\_date,

f.film\_id,

c.name AS category\_name,

r.customer\_id,

p.amount AS payment\_amount

FROM

rental r

JOIN

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

JOIN

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

JOIN

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

JOIN

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

JOIN

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

-- Repopulate the summary table by using ON CONFLICT to avoid duplicates

INSERT INTO summary\_rental\_report (month, category\_name, total\_rentals, total\_revenue)

SELECT

extract\_month(r.rental\_date) AS month,

c.name AS category\_name,

COUNT(\*) AS total\_rentals,

SUM(p.amount) AS total\_revenue

FROM

rental r

JOIN

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

JOIN

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

JOIN

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

JOIN

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

JOIN

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

GROUP BY

month, category\_name

ON CONFLICT (month, category\_name)

DO UPDATE SET

total\_rentals = EXCLUDED.total\_rentals,

total\_revenue = EXCLUDED.total\_revenue;

END;

$$ LANGUAGE plpgsql;

Part F1

**Job Scheduling Tool:**

* A relevant job scheduling tool to automate this stored procedure would be **pg\_cron**, which allows for scheduling tasks like refreshing the report at regular intervals (e.g., monthly).

Part H

No sources were used to support my submission.