**Data Analysis**

Mohammad Saleh

WGU

CSN1: Advanced Data Management

6/14/2024

1. 1. Fields to include in the detailed table.

Monthly DVD Rental Revenue and customer activity report

below is the list of the specific fields to include in the detailed table and summary table

**Detailed Table**

* Rental ID
* Rental Date
* Return Date
* Customer ID
* Customer Name
* Film Title
* Category
* Rental Duration
* Rental Fee

**Summary table**

* Total Rentals
* Total Revenue
* Average Rental Duration
* Most Popular Film
* Most Popular Category
* Active Customers (number of customers who rented at least one DVD)

**2. Below is a list of the type of data fields used for this report**

* **Integer**: Rental ID, Customer ID, Total Rentals, Active Customers
* **Date/Time**: Rental Date, Return Date
* **Text/String**: Customer Name, Film Title, Category, Most Popular Film, Most Popular Category
* **Numeric**: Rental Duration (calculated in days), Rental Fee, Total Revenue, Average Rental Duration

3. **Specific tables from the given dataset to provide the data necessary for detailed table section and summary table.**

* **rental**: This table provides details on each DVD rental, including Rental ID, Rental Date, Return Date, and Customer ID.
* **customer**: This table provides customer details, including Customer ID and Customer Name.
* **film**: This table provides film details, including Film Title and Category.

4. **Customer Transformation with a User-Defined Function**

**Field: Rental Duration**

**Transformation Explanation**: The Rental Duration will be calculated as Return Date - Rental Date. This has to be handled by a custom function to properly return the duration in terms of days and also handle where the return date may be null (meaning the DVD has not been returned yet).

**5. Business Uses of the Detailed Table and Summary Table**

**Detailed Table:**

**Use:** It has an extensive list that contains information concerning individual rental occurrences. As a result, there is the possibility of exploring overall tendencies in renting behaviours and detect individual customer activities as well as control stock by depicting the demand of such films and genres.

**Summary Table:**

**Use**: The summary table provides an overview of the macro environment analysis of the rental business. It assists stakeholders in making sound decisions regarding the status of the business by providing such aspects as the total revenue, average rental period, and other customer related statistics. It is useful in decision-making processes like choosing marketing promotions for best selling categories or films and vice versa, and decision making about the need to improve on.

#### **Report Refresh Frequency**

**Refresh Frequency**: Monthly

**Explanation**: Considering that this report is made to present rental revenue and customer activity on a monthly basis, it should be updated on a monthly basis. This frequency is beneficial for the stakeholders as it means they have the most recent data on the company’s performance for the most recent month.

1. **Original code for the custom function that calculates the rental duration which was identified in part A4**

CREATE OR REPLACE FUNCTION calculate\_rental\_duration(rental\_date DATE, return\_date DATE)

RETURNS INTEGER AS $$

BEGIN

IF return\_date IS NULL THEN

RETURN CURRENT\_DATE - rental\_date;

ELSE

RETURN return\_date - rental\_date;

END IF;

END;

$$ LANGUAGE plpgsql;

1. **Below is the SQL code to create a detailed and summary table to hold the necessary information**

**Detailed table**

CREATE TABLE detailed\_rental\_report (

rental\_id INTEGER PRIMARY KEY,

rental\_date DATE NOT NULL,

return\_date DATE,

customer\_id INTEGER NOT NULL,

customer\_name VARCHAR(255) NOT NULL,

film\_title VARCHAR(255) NOT NULL,

category VARCHAR(255) NOT NULL,

rental\_duration INTEGER,

rental\_fee NUMERIC(5,2) NOT NULL

);

**SQL code to create summary table**

CREATE TABLE summary\_rental\_report (

report\_month DATE PRIMARY KEY,

total\_rentals INTEGER NOT NULL,

total\_revenue NUMERIC(10,2) NOT NULL,

average\_rental\_duration NUMERIC(5,2) NOT NULL,

most\_popular\_film VARCHAR(255) NOT NULL,

most\_popular\_category VARCHAR(255) NOT NULL,

active\_customers INTEGER NOT NULL

);

**SQL code to insert Data into detailed table**

INSERT INTO detailed\_rental\_report (rental\_id, rental\_date, return\_date, customer\_id, customer\_name, film\_title, category, rental\_duration, rental\_fee)

SELECT

r.rental\_id,

r.rental\_date,

r.return\_date,

c.customer\_id,

c.first\_name || ' ' || c.last\_name AS customer\_name,

f.title AS film\_title,

f.category,

calculate\_rental\_duration(r.rental\_date, r.return\_date) AS rental\_duration,

r.rental\_rate AS rental\_fee

FROM

rental r

JOIN

customer c ON r.customer\_id = c.customer\_id

JOIN

film f ON r.film\_id = f.film\_id;

**SQL code to insert data into summary table**

INSERT INTO summary\_rental\_report (report\_month, total\_rentals, total\_revenue, average\_rental\_duration, most\_popular\_film, most\_popular\_category, active\_customers)

SELECT

DATE\_TRUNC('month', r.rental\_date) AS report\_month,

COUNT(r.rental\_id) AS total\_rentals,

SUM(r.rental\_rate) AS total\_revenue,

AVG(calculate\_rental\_duration(r.rental\_date, r.return\_date)) AS average\_rental\_duration,

(

SELECT f.title

FROM rental r2

JOIN film f ON r2.film\_id = f.film\_id

GROUP BY f.title

ORDER BY COUNT(r2.rental\_id) DESC

LIMIT 1

) AS most\_popular\_film,

(

SELECT f.category

FROM rental r2

JOIN film f ON r2.film\_id = f.film\_id

GROUP BY f.category

ORDER BY COUNT(r2.rental\_id) DESC

LIMIT 1

) AS most\_popular\_category,

COUNT(DISTINCT r.customer\_id) AS active\_customers

FROM

rental r

GROUP BY

report\_month;

1. **Below is the SQL query to extract the raw data needed for the detailed section of the report from the source database.**

SELECT

r.rental\_id,

r.rental\_date,

r.return\_date,

c.customer\_id,

c.first\_name || ' ' || c.last\_name AS customer\_name,

f.title AS film\_title,

cat.name AS category,

calculate\_rental\_duration(r.rental\_date, r.return\_date) AS rental\_duration,

r.rental\_rate AS rental\_fee

FROM

rental r

JOIN

customer c ON r.customer\_id = c.customer\_id

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

1. **Below is the SQL code to create a trigger on the detailed table that will continually update the summary table as data is added to the detailed table.**

CREATE OR REPLACE FUNCTION update\_summary\_rental\_report()

RETURNS TRIGGER AS $$

BEGIN

-- Update total rentals and total revenue

UPDATE summary\_rental\_report

SET

total\_rentals = total\_rentals + 1,

total\_revenue = total\_revenue + NEW.rental\_fee,

average\_rental\_duration = (

(average\_rental\_duration \* (total\_rentals - 1) + NEW.rental\_duration) / total\_rentals

),

active\_customers = (

SELECT COUNT(DISTINCT customer\_id)

FROM detailed\_rental\_report

WHERE DATE\_TRUNC('month', rental\_date) = DATE\_TRUNC('month', NEW.rental\_date)

)

WHERE

report\_month = DATE\_TRUNC('month', NEW.rental\_date);

-- Insert new month record if not exists

IF NOT FOUND THEN

INSERT INTO summary\_rental\_report (

report\_month, total\_rentals, total\_revenue, average\_rental\_duration, most\_popular\_film, most\_popular\_category, active\_customers

)

VALUES (

DATE\_TRUNC('month', NEW.rental\_date), 1, NEW.rental\_fee, NEW.rental\_duration, '', '', 1

);

END IF;

-- Update most popular film and category

UPDATE summary\_rental\_report

SET

most\_popular\_film = (

SELECT f.title

FROM detailed\_rental\_report dr

JOIN film f ON dr.film\_title = f.title

WHERE DATE\_TRUNC('month', dr.rental\_date) = report\_month

GROUP BY f.title

ORDER BY COUNT(dr.rental\_id) DESC

LIMIT 1

),

most\_popular\_category = (

SELECT c.name

FROM detailed\_rental\_report dr

JOIN film f ON dr.film\_title = f.title

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

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

WHERE DATE\_TRUNC('month', dr.rental\_date) = report\_month

GROUP BY c.name

ORDER BY COUNT(dr.rental\_id) DESC

LIMIT 1

)

WHERE

report\_month = DATE\_TRUNC('month', NEW.rental\_date);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

Below is the trigger that will call the above function whenever a new row is inserted into the detailed table

CREATE TRIGGER update\_summary\_trigger

AFTER INSERT ON detailed\_rental\_report

FOR EACH ROW

EXECUTE FUNCTION update\_summary\_rental\_report();

1. **Below is the SQL code for a stored procedure that refreshes the data in both the detailed and summary table.**

CREATE OR REPLACE PROCEDURE refresh\_rental\_report()

LANGUAGE plpgsql

AS $$

BEGIN

-- Clear contents of the detailed table

TRUNCATE TABLE detailed\_rental\_report;

-- Clear contents of the summary table

TRUNCATE TABLE summary\_rental\_report;

-- Insert new data into the detailed table

INSERT INTO detailed\_rental\_report (rental\_id, rental\_date, return\_date, customer\_id, customer\_name, film\_title, category, rental\_duration, rental\_fee)

SELECT

r.rental\_id,

r.rental\_date,

r.return\_date,

c.customer\_id,

c.first\_name || ' ' || c.last\_name AS customer\_name,

f.title AS film\_title,

cat.name AS category,

calculate\_rental\_duration(r.rental\_date, r.return\_date) AS rental\_duration,

r.rental\_rate AS rental\_fee

FROM

rental r

JOIN

customer c ON r.customer\_id = c.customer\_id

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

-- Insert new data into the summary table

INSERT INTO summary\_rental\_report (report\_month, total\_rentals, total\_revenue, average\_rental\_duration, most\_popular\_film, most\_popular\_category, active\_customers)

SELECT

DATE\_TRUNC('month', r.rental\_date) AS report\_month,

COUNT(r.rental\_id) AS total\_rentals,

SUM(r.rental\_rate) AS total\_revenue,

AVG(calculate\_rental\_duration(r.rental\_date, r.return\_date)) AS average\_rental\_duration,

(

SELECT f.title

FROM rental r2

JOIN film f ON r2.film\_id = f.film\_id

GROUP BY f.title

ORDER BY COUNT(r2.rental\_id) DESC

LIMIT 1

) AS most\_popular\_film,

(

SELECT c.name

FROM rental r2

JOIN film f ON r2.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

GROUP BY c.name

ORDER BY COUNT(r2.rental\_id) DESC

LIMIT 1

) AS most\_popular\_category,

COUNT(DISTINCT r.customer\_id) AS active\_customers

FROM

rental r

GROUP BY

report\_month;

END;

$$;

### **Job Scheduling Tool: pg\_cron**

A relevant job scheduling tool that can be used to automate the execution of this stored procedure is ***pg\_cron. pg\_cron*** is a PostgreSQL extension for running periodic jobs. It allows you to schedule PostgreSQL commands directly from the database.