**Below is the transcript for video recording demonstrating the functionality of the code used for the analysis.**

**Monthly DVD Rental Revenue and Customer Activity Report**

Hello, and welcome to this demonstration on generating a Monthly DVD Rental Revenue and Customer Activity Report using PostgreSQL. Today, we'll walk through the process of setting up the necessary tables, creating functions and triggers, and running a stored procedure to refresh our report data. Let's get started!

**In this presentation we will discuss the following:**

* Introduction to the DVD Rental Database
* Creating the Detailed and Summary Tables
* Custom Function for Rental Duration
* Inserting Data into the Tables
* Creating and Using a Trigger
* Refreshing the Report Data
* Automating the Process with pg\_cron

**Introduction to the DVD Rental Database**

Our DVD Rental Database contains tables for rentals, customers, films, and categories. We will use this data to generate detailed and summary reports.

**Now let us create a detailed and summary tables**

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

);

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

);

**Next we need a function to calculate the rental duration. This function will handle cases where the return date might be null.**

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;

Now let us insert data into our detailed table using a query that joins multiple tables and uses our custom function.

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;

**To keep our summary table updated, we create a trigger that updates it whenever a new row is inserted into 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;

CREATE TRIGGER update\_summary\_trigger

AFTER INSERT ON detailed\_rental\_report

FOR EACH ROW

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

**To refresh our report data, we'll use a stored procedure that clears both tables and re-populates them.**

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;

$$;

**Finally, we can automate this process using *pg\_cron*, a PostgreSQL extension for scheduling jobs.**

-- Load the pg\_cron extension

CREATE EXTENSION pg\_cron;

-- Schedule the stored procedure to run at midnight on the first day of every month

SELECT cron.schedule('0 0 1 \* \*', $$CALL refresh\_rental\_report();$$);

Now with that we have successfully set up a process to generate and refresh our monthly DVD rental revenue and customer activity report. Thank you for watching this demonstration.