Course ID: D326
Course title: Advanced Data Management
Project Title: Top 25 rentals
Student name: Lawrence Sanzogni
Student ID: 012143472

Introduction

A common goal among business models is to increase sales. Thanks to current technology, we can create sales strategies more quickly than ever using big data and analytics to analyze sales trends and potentially predict growth areas. This will be a contained demonstration of these techniques and principles, using a dvd rental store as our platform.

All customer data - if managed carefully - is stored, organized, and archived in a storage system that can be accessed. Modern examples of storage systems - such as the one in this report - are relational databases. This demonstration will include database queries used to properly support sales proposals.

The full presentation will be made up of two SQL tables: a detailed table, containing granular information, and a summary table.

Sections A1 - A6

A typical metric of a business's health is a list of top products. This report will generate the top 25 most rented DVD titles in the past 6 months.

The summary table will include a film title field (plain text), for each film name included in the top 25 list, and a total profits field (a numeric value) which is the sum of each film's profits in the past 6 months:

```
f.title as top_25_rentals (varchar 255),
Total_profits_6_months (numeric)
```

The fields in the top 25 detailed table include: a film id (an integer) which is a unique identifier for each film in the database a film title (plain text),a rental id (an integer) which represents each individual rental transaction of that movie, a rental date (a rental date that has its original timestamp removed through a custom transformation using a UDF for better legibility by removing the timestamps from the date, the scope of data doesn't require time of day for each transaction; e.g., 2005-05-24 22:54:33 -> 2005-05-24), and the payment amount for each individual rental (a numeric value with two decimal placements).

```
film_id: integer,
film_title: plain text,
rental_id: custom_function(date_with_timestamp) -> date // <- field that
```

```
requires custom transformation,
rental_date: date,
rental_amount: numeric (5,2)
```

Data from the following tables from the rental database are queried to build the report tables: *‘film’*, *‘inventory’*, *‘rental’*, and *‘payment’*.

The summary table provides a macro view of current business performance while the detailed table can zoom in on individual, day-to-day performance; as well as explain how the data from the summary table was collected and calculated. The purpose of both tables is to have a snapshot of business performance that spans 6 months. A bi-annual report can be useful in determining future purchase trends by examining past ones. A monthly refresh rate will help keep data current.

Section B

This is the user-defined function that performs the custom transformation on the timestamp values into dates:

```
-- CUSTOM TRANSFORMATION FUNCTION: removes timestamp from dates
create or replace function remove_timestamp(x timestamp) returns date as $$
      declare
      new_date date;
      begin
      select
      date(x)
      into new_date;
      return new_date;
      end;
$$language plpgsql;
```

Section C

These are the queries that create the summary and detailed tables:

```
-- SUMMARY TABLE
create table top_25_summary (
      film_title varchar(255)not null,
      total_profits_6_months numeric not null
);
```

```
-- DETAILED TABLE
create table top_25_detailed (
      film_id integer not null,
      film_title varchar(255) not null,
      rent_id integer not null,
      rent_date date not null,
      rent_amount numeric(5,2) not null
);
```

Section D:

This is the query to extract the data to populate the detailed table:

```
-- DETAILED TABLE UPDATE
insert into top_25_detailed (film_id, film_title, rent_id, rent_date,
rent_amount)
select f.film_id, f.title, r.rental_id, remove_timestamp(r.rental_date),
p.amount
from film f
left join inventory i
on f.film_id=i.film_id
left join rental r
on i.inventory_id=r.inventory_id
inner join payment p
on r.rental_id=p.rental_id
where r.rental_date
between '2005-08-14' and '2006-02-14';
```

Section E:

This is the trigger on the detailed table that will update the summary table as data is added to the detailed table.

```
-- TRIGGER FUNCTION TO UPDATE SUMMARY
create or replace function update_summary() returns trigger as $$
      begin
            -- flush summary table
            truncate top_25_summary;

            -- update summary table
            insert into top_25_summary(film_title, total_profits_6_months)
            select film_title, sum(rent_amount)
            from top_25_detailed
            group by film_title
            order by sum(rent_amount) desc
            limit 25;
            return null;
      end;
$$ language plpgsql;

-- TRIGGER TO UPDATE SUMMARY
create or replace trigger update_summary after insert or update
      on top_25_detailed
      for each statement
      execute function update_summary();
```

Section F:

This is the procedure to clear and update both tables

```
-- PROCEDURE: empty and update detailed and summary tables
create or replace procedure refresh_detailed_and_summary_tables() language
plpgsql as $$

      Begin
            -- flush both tables
            truncate table top_25_detailed, top_25_summary;

            -- populate detailed table & summary table (table is populated by
trigger
            insert into top_25_detailed(film_id, film_title, rent_id,
rent_date, rent_amount)
                  select f.film_id, f.title, r.rental_id,
remove_timestamp(r.rental_date), p.amount
                  from film f
                  left join inventory i
                  on f.film_id=i.film_id
                  left join rental r
                  on i.inventory_id=r.inventory_id
                  inner join payment p
                  on r.rental_id=p.rental_id
                  where r.rental_date
                  between '2005-08-14' and '2006-02-14';

      end;
$$
```

A couple of job scheduling tools that can help automate updating the tables every month are:

- *pg_cron* :a cron based scheduler that combines cron and postgresql syntax (Pg_Cron Github, 2024),
- *pgagent :* a Postgres extension you can run in the PgAdmin4 GUI.

Since neither program can be easily installed, set up, or run on Mac OS (the operating system used for this assignment), this report won't include a working example.

For a high level example, here is a theoretical script that could be written using *pg_cron* and ran inside the psql CLI environment for this report:

```
SELECT cron.schedule('0 0 1 * *', $$call refresh_detailed_and_summary_tables()$$);
schedule
```

This scheduler executes the stored procedure to refresh the detailed and summary tables at midnight on the first of every month (Crontab Guru, 2025).

References:

- **https://github.com/citusdata/pg_cron?tab=readme-ov-file**
- **https://crontab.guru/#0_0_1_*_***