Section A

One question we may have answered using the DVD Rental Database is “Which actor has brought in the most money for the business?” We will need to extract several pieces of data from the database to answer this question. We will need an Integer called, film_id that is uniquely attributed to a specific film,title, actor_id that will be uniquely attributed integer to a specific actor, a rental_rate that will be the specific rental fee in the form of a numeric for a specific film, inventory_id, an integer unique to the item inventoried and the rental_id, another integer specific to one specific rental transaction. All of this data can be gathered from the film film_actor, actor, inventory and rental tables. Inside my vals table I will include columns for film_id, actor_id, rental_rate inventory_id and rental_id. Inside my endResults table I will have columns for the actor_id, the sum of rental_id’s pertaining to each specific film and the total rental income of all rental transactions involving the films related to that specific entity. Data transformations that I will be utilizing will be the count and summation of the films and rental fees. Displaying these in descending order will allow myself as the user to see what actor_id brings in the most income through rental pricing and amount of rentals for the business. The business use for this could be to study the trends in film rentals from the previous week to find which actor is bringing in the most profit for the rental business. The business could then make decisions on who and what films they should focus on promoting more to help drive profits higher. This report could be run on a quarterly basis as with the release of new movies popularity in one actor or film may change who might the rental company be able to expect more of a profit from.

Section B

```
DROP TABLE IF EXISTS vals;
CREATE TABLE vals(
        film_id integer,
        actor_id integer,
        rental_rate numeric(5,2),
        inventory_id integer,
        rental_id integer);


DROP TABLE IF EXISTS endResults;
CREATE TABLE endResults(
        rentalTotal numeric (8,2),
        actor_id integer,
        tranSum integer
        );
```

section C

```
INSERT INTO vals (film_id, rental_rate, actor_id, inventory_id, rental_id)
SELECT c.film_id, c.rental_rate, d.actor_id, e.inventory_id, f.rental_id
FROM film c
INNER JOIN film_actor d ON c.film_id = d.film_id
INNER JOIN inventory e ON c.film_id = e.film_id
```

```
INNER JOIN rental f ON e.inventory_id = f.inventory_id
ORDER BY actor_id;
```

Section D

```
DROP FUNCTION erValues();
CREATE FUNCTION erValues()
RETURNS TRIGGER AS $$
BEGIN
DELETE FROM endResults;
INSERT INTO endResults(
        SELECT
                count(DISTINCT rental_id), actor_id,sum(rental_rate)
        FROM vals
        GROUP BY actor_id
        ORDER BY actor_id);
RETURN NEW;
END; $$ LANGUAGE PLPGSQL;
```

Section E

```
CREATE TRIGGER er_refresh
AFTER INSERT ON vals
FOR EACH STATEMENT
EXECUTE PROCEDURE erValues();
```

```
SELECT * FROM vals;
INSERT INTO vals VALUES( 166, 1, 2.99, 758, 4608);
SELECT * FROM endResults ORDER BY rentalTotal DESC;
```

Section F

This section creates the stored procedure that will update the vals table and the endResults table. This could be ran at the end of every quarter or month to allow tracking of current trends as well as allow auditing of accuracy of the tables. First one would install pgAdmin 4 using the command "$ sudo apt install pgadmin4 pgadmin4-apache"(Hugo Dias, An overview of job scheduling tools for postgresql 2020). After making sure one has installed pgAdmin 4, they would need to make sure the plpgsql has been defined (Hugo Dias, An overview of job scheduling tools for postgresql 2020). Lastly, an install of the pgAgent is required using the command "$ sudo apt-get install pgagent" (Hugo Dias, An overview of job scheduling tools for postgresql 2020). In the last step we have to create the pgAgent extension inserting "CREATE EXTENSION pageant". This will create the tables and functions needed for the pgAgent operation. Inside the pgAgent is where one can then select the create button under the pgAgents jobs tab while under the schedules tab we can define the function and how frequently it is ran. (Hugo Dias, An overview of job scheduling tools for postgresql 2020). To be able to have the agent running in the background one will have to manually launch the following process "/usr/bin/pgagent host=localhost dbname=postgres user=postgres port=5432 -l 1" (Hugo Dias, An overview of job scheduling tools for postgresql 2020).

```
CREATE PROCEDURE refresh_er()
LANGUAGE PLPGSQL
AS $$
BEGIN
DELETE FROM vals;
INSERT INTO vals (film_id, rental_rate, actor_id, inventory_id, rental_id)
SELECT c.film_id, c.rental_rate, d.actor_id, e.inventory_id, f.rental_id
FROM film c
INNER JOIN film_actor d ON c.film_id = d.film_id
INNER JOIN inventory e ON c.film_id = e.film_id
INNER JOIN rental f ON e.inventory_id = f.inventory_id
ORDER BY actor_id;

END; $$;

CALL refresh_er();
```

Sources

Dias, H. (2020, February 3). *An overview of job scheduling tools for postgresql*. Severalnines. Retrieved April 28th, 2022, from https://severalnines.com/database-blog/overview-job-scheduling-tools-postgresql

Used to provide information on pgAgent for scheduled tasks as well as understanding the use of PLPGSQL for creating a procedure and trigger.

PostgreSQL create trigger. PostgreSQL Tutorial. (n.d.). Retrieved May 3, 2022, from https://www.postgresqltutorial.com/postgresql-triggers/creating-first-trigger-postgresql/

Used to understand exactly how to create a trigger for a function

PostgreSQL inner join. PostgreSQL Tutorial. (n.d.). Retrieved May 3, 2022, from https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-inner-join/

Used to better understand how to write Inner Joins using postgreSQL