Alireza Barzegar
ID #010057350
WGU Email: abarzeg@wgu.edu
06/21/2024

A) This project aims to create a business report that answers a business question for the DVD rental business, using the available information from the business's database. The business question is, what are the top 10 actors whose movies have been rented the most? Answering this will help the business recognize which actors are more popular among their customers, so they can buy movies featuring those actors to rent out and increase revenue.

A1) The detailed table will have three fields actor_id, full_name, and payment_id. The summary table will have three fields actor_id, full_name, and movies_rented.

A2) Detailed table field data types and definitions: Actor_id is an integer and the primary key in the actor table, containing unique values identifying a single actor. Payment_id is an integer and the primary key in the payment table, with each unique value identifying a single payment. Full_name is the concatenation of the first_name and last name columns of the actor table, specifying the full name of an actor.

Summary table field data types and definitions: Similar to the detailed table, Actor_id is an integer that identifies a single actor in the table. Full_name is also the first_name and last_name of the actor concatenated together to represent the actor's full name. Movies_rented is an integer and shows the total number of rented movies for an actor.

A3) Payment and actor tables from the dataset were used to provide the necessary data for detailed and summary tables.

A4) the full_name column of the detailed table will require a user-defined function to concatenate first_name and last_name. This allows for simplicity and ease of use since we'll need the full name of an actor querying for them or displaying their information. Therefore, having them as one column instead of two is more efficient.

A5) A stakeholder could find out how much revenue each actor's movies are bringing in for the DVD rental business, by using the information from the detailed table to query the payment table. Then, they can prioritize stocking movies for actors who have brought in the most revenue. The summary table's information tells the top 10 actors whose movies are rented the

most. A stakeholder can use this information when running ads for their rental service to display movie pictures of characters played by actors on the summary table since they're more likely to attract people.

A6) The report should be refreshed monthly to keep the ads updated with what's popular with their customers.

B)

```
CREATE OR REPLACE FUNCTION full_name(first_name VARCHAR(200), last_name
VARCHAR(200))
	RETURNS VARCHAR(400)
	LANGUAGE plpgsql
AS
$$
DECLARE full_name VARCHAR(400);
BEGIN

SELECT first_name || ' ' || last_name INTO full_name;
RETURN full_name;

END;
$$;
```

C)

```
CREATE TABLE detailed(
	actor_id INTEGER NOT NULL,
	full_name VARCHAR(400),
	payment_id INTEGER NOT NULL
)



CREATE TABLE summary(
	actor_id INTEGER NOT NULL,
	full_name VARCHAR(400) NOT NULL,
	movies_rented INTEGER NOT NULL
)
```

D)

```
INSERT INTO detailed(actor_id, full_name, payment_id)
SELECT actor.actor_id, full_name(first_name, last_name), payment_id
FROM actor
INNER JOIN film_actor
ON actor.actor_id = film_actor.actor_id
INNER JOIN inventory
ON film_actor.film_id = inventory.film_id
INNER JOIN rental
ON inventory.inventory_id = rental.inventory_id
INNER JOIN payment
ON rental.rental_id = payment.rental_id;
```

E)

```
CREATE OR REPLACE FUNCTION fill_summary()
        RETURNS TRIGGER
        LANGUAGE plpgsql
AS
$$
BEGIN

INSERT INTO summary
SELECT actor_id, full_name, COUNT(payment_id)
FROM detailed
GROUP BY actor_id, full_name
ORDER BY COUNT(payment_id) DESC
LIMIT 10;

RETURN NEW;

END;
$$;

CREATE TRIGGER refill_summary
        AFTER INSERT
        ON detailed
        FOR EACH STATEMENT
```

```
    EXECUTE PROCEDURE fill_summary();
```

F)

```
CREATE OR REPLACE PROCEDURE refresh_tables()
    LANGUAGE plpgsql
AS
$$
BEGIN

DELETE FROM detailed;
DELETE FROM summary;

INSERT INTO detailed(actor_id, full_name, payment_id)
SELECT actor.actor_id, full_name(first_name, last_name), payment_id
FROM actor
INNER JOIN film_actor
ON actor.actor_id = film_actor.actor_id
INNER JOIN inventory
ON film_actor.film_id = inventory.film_id
INNER JOIN rental
ON inventory.inventory_id = rental.inventory_id
INNER JOIN payment
ON rental.rental_id = payment.rental_id;

RETURN;
END;
$$
```

F1) A job scheduling tool that can be used to automate the stored procedure is pg_cron. This extension is native to PostgreSQL, making its setup easy. It is also suitable for scheduling simple periodic jobs, such as the stored procedure for refreshing the detailed and summary tables, which needs to run monthly.