D191 Data Report

Jeremy Bedsaul
Student ID # 001460177

Introduction

The following report was prepared to share insights into the rental revenue generated by film genre over a specified time period. Included in the Detailed Section will be specifics about each payment made, including the date, film title, genre, and the amount of revenue. The Summary Section focuses on aggregating the total revenue generated by film genre.

This report gives us insight into the top 5 highest grossing genres of films based on total revenue collected.

Section A:

1. Data Used in this Report:
   - Rental details payments by film.
   - Payment details: date and amount.
   - Inventory data to see what films were rented during this time frame.
   - Film details: title and associated genre. ◦
   Genre details by category.
2. Tables Used:
   - Detailed Section: 'payment, rental', 'inventory', 'film', 'film_category', and 'category'. ◦
   Summary Section: 'detailed_report' (aggregated data will be pulled from here)
3. Fields Used:
   - Detailed Section: 'payment_date', 'film_title', 'genre_name', and 'amount'
   - Summary Section: 'genre_name' and 'total_amount'
4. Custom Transformation:
   - In this report the field 'genre_name' was used for a custom transformation. The specific business use for the transformation was simulating a change in the genre name of 'Animation' to a new name of 'Animated'. This type of change may be warranted for branding reasons or to aid in clearer understanding.
5. Possible Business Use Cases:
   - Detailed Section: This section of the data could be used by a variety of business units. One possible use case would be for finance and audit teams in validating individual transactions and to investigate any outliers or discrepancies.
   - Summary Section: This section would be best utilized by executives and other decision makers. Since this provides a high-level overview, the data can be scanned quickly to gain insights into various aspects of the business such as inventory purchases, as well as future investments into different genres.

6. Report Refresh Frequency:
   - With this report being based on rental trends, and given that new rentals will be occurring daily, the report should be refreshed daily to gain the most up to date insights possible. Though this report could be adapted just as easily for weekly, monthly, quarterly, or annual trends as well.

Section B:

```
CREATE TABLE detailed_report
        ( report_id SERIAL PRIMARY KEY,
        payment_date TIMESTAMP WITHOUT TIME ZONE,
        film_title VARCHAR(255),
        genre_name VARCHAR(255),
        amount NUMERIC(10,2)
);

CREATE TABLE summary_report
        ( genre_name VARCHAR(255),
        total_amount NUMERIC(10,2),
        PRIMARY KEY (genre_name)
);
```

Section D:

```
CREATE OR REPLACE FUNCTION transform_genre_name() RETURNS TRIGGER AS $$
BEGIN
        IF NEW.genre_name = 'Animation' THEN
                NEW.genre_name := 'Animated';
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

--applying transformation

CREATE TRIGGER transform_genre_name_trigger
BEFORE INSERT OR UPDATE ON detailed_report
FOR EACH ROW
EXECUTE FUNCTION transform_genre_name();
```

Section E:

```
CREATE OR REPLACE FUNCTION update_summary() RETURNS TRIGGER AS $$
BEGIN
        INSERT INTO summary_report(genre_name, total_amount)
        VALUES (NEW.genre_name, NEW.amount)
        ON CONFLICT(genre_name)
        DO UPDATE SET total_amount = summary_report.total_amount + NEW.amount;
RETURN NEW; END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_summary_trigger
AFTER INSERT ON detailed_report
FOR EACH ROW
EXECUTE FUNCTION update_summary();
```

Section C:

```
INSERT INTO detailed_report(payment_date, film_title, genre_name, amount)
SELECT
        p.payment_date::TIMESTAMP WITHOUT TIME ZONE,
        f.title,
        c.name,
        p.amount
FROM
        payment p
JOIN
        rental r ON p.rental_id = r.rental_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 c ON fc.category_id = c.category_id;
SELECT * FROM detailed_report;
```

Section F:

```
CREATE OR REPLACE PROCEDURE refresh_data() AS $$
```

```
BEGIN
	TRUNCATE TABLE detailed_report, summary_report;
	INSERT INTO detailed_report(payment_date, film_title, genre_name, amount)
	SELECT
		p.payment_date::TIMESTAMP WITHOUT TIME ZONE,
		f.title,
		c.name,
		p.amount
	FROM
		payment p
	JOIN
		rental r ON p.rental_id = r.rental_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 c ON fc.category_id = c.category_id;


	INSERT INTO summary_report (genre_name, total_amount)
	SELECT genre_name, SUM(amount)
	FROM detailed_report
	GROUP BY genre_name
	ON CONFLICT(genre_name)
	DO UPDATE SET total_amount = EXCLUDED.total_amount;
END;
$$ LANGUAGE plpgsql;

CALL refresh_data();
SELECT * FROM detailed_report;
SELECT * FROM summary_report;
```

Section F1:

- To ensure data freshness, I would recommend running the stored procedure 'refresh_data()' daily. There are multiple ways this could be accomplished. For this, I would suggest either scheduling it in PostgreSQL's built in scheduler, 'pg_cron' or by using an external

scheduler. If using ‘pg_cron’ we could add a job to run the code ‘CALL refresh_data();’ at the suggested frequency.

Final Summary:

- The top five highest grossing genres are:
  1. Sports
  2. Sci-Fi
  3. Animated
  4. Drama
  5. Comedy

References:

- Dias, H. (2020, February 3). *An overview of scheduling tools for PostgreSQL*. Severalnines. Retrieved August 18, 2020, from https://severalnines.com/database-blog/overview-job-scheduling-toolspostgresql
- Ravi, J. (2023). *Advanced PostgreSQL* [Video]. LinkedIn Learning. https://www.linkedin.com/learning/advanced-postgresql/advanced-features-inpostgresql?resume=false&u=2045532