DVD RENTAL DATA

A. Real-World Business Report Summary

The following document contains queries, code, and use-case scenarios. Rather than creating a report of total sales, customers, or rentals, I wanted to explore where the business may be able to make some changes to increase profits and customer satisfaction. Outlined below is the code to create a month-over-month report that will show the average sale per customer, the two top selling film categories, and the least selling film category. In analyzing these areas, we can better utilize shelfspace and shift inventory to better fit the demand of our customer base.

1. Specific Fields for Detailed and Summary Tables

Detailed Table Fields:

month_start_date: Formatted start date of the month.
rental_id: Unique identifier for each rental transaction.
customer_id: Unique identifier for each customer.
film_id: Unique identifier for each film.
category_id: Unique identifier for each film category.
category_name: Name of the film category.
rental_date: Date when the rental occurred.
return_date: Date when the rental was returned.
amount: Amount paid for the rental.

Summary Table Fields:

month_start_date: Formatted start date of the month.
total_rentals: Total number of rentals in the month.
average_spend: Average amount spent by customers in the month.
top_genre_id: Unique identifier for the top rented genre.
top_genre: Name of the top rented genre.
top_genre_count: Count of rentals for the top rented genre.
second_top_genre_id: Unique identifier for the second most rented genre.
second_top_genre: Name of the second most rented genre.
second_top_genre_count: Count of rentals for the second most rented genre.
least_rented_genre_id: Unique identifier for the least rented genre.
least_rented_genre: Name of the least rented genre.
least_rented_genre_count: Count of rentals for the least rented genre.

2. Types of Data Fields Used

Date/Time Fields: month_start_date, rental_date, return_date
Numeric Fields: rental_id, customer_id, film_id, category_id, amount, total_rentals, top_genre_count, second_top_genre_count, least_rented_genre_count
Text Fields: category_name, top_genre, second_top_genre, least_rented_genre
Currency Field: average_spend

3. Source Tables for Data

Detailed Table:

rental: Provides data for rental_id, customer_id, rental_date, return_date.
payment: Provides data for amount.
inventory: Links rental to film.
film: Provides data for film_id.
film_category: Links film to category.
category: Provides data for category_id and category_name.

Summary Table:

rental
payment
inventory
film
film_category
Category

4. Custom Transformation Field

Field: month_start_date
Transformation: The month_start_date field requires a custom transformation to display the full name of the month. This transformation uses a user-defined function format_date which formats the date to show only the full name of the month (e.g., "January").

5. Business Uses of the Detailed and Summary Tables

Detailed Table:
Operational Use: Track daily rental activities, monitor individual customer transactions, and analyze rental patterns on a detailed level.
Customer Insights: Identify customer preferences, frequent renters, and peak rental times.

Summary Table:
Strategic Use: Understand overall rental trends, assess the financial performance of rental operations, and make data-driven decisions for inventory management.
Genre Popularity: Identify the most and least popular genres to inform decisions on phasing out less popular genres and stocking more of the popular ones.

6. Report Refresh Frequency

The report should be refreshed monthly to ensure it remains relevant and useful for stakeholders. This frequency aligns with the aggregation level of the data (monthly) and provides timely insights for business planning and strategy adjustments.

B. Code for Transformation Function

The identified transformation is to format the month_start_date to display the year followed by the full name of the month. This will be easier to read than the standard YYYY-MM-DD output.

```
CREATE OR REPLACE FUNCTION format_year_month(input_date DATE)
RETURNS TEXT AS $$
BEGIN
	RETURN TO_CHAR(input_date, 'YYYY Month');
END;
$$ LANGUAGE plpgsql;
```

C. SQL Code to Create Detailed and Summary Tables

Detailed Table Creation

```
CREATE TABLE monthly_detailed (
	month_start_date TEXT,
	rental_id INTEGER,
	customer_id INTEGER,
	film_id INTEGER,
	category_id INTEGER,
	category_name TEXT,
	rental_date DATE,
	return_date DATE,
	amount NUMERIC
);
```

Summary Table Creation

```
CREATE TABLE monthly_summary (
	month_start_date TEXT,
	total_rentals INTEGER,
	average_spend TEXT,
	top_genre_id INTEGER,
	top_genre TEXT,
	top_genre_count INTEGER,
	second_top_genre_id INTEGER,
	second_top_genre TEXT,
	second_top_genre_count INTEGER,
	least_rented_genre_id INTEGER,
	least_rented_genre TEXT,
	least_rented_genre_count INTEGER
);
```

D. Query to extract raw data

```
WITH DetailedRentals AS (
	SELECT
		format_year_month(DATE_TRUNC('month', r.rental_date)) AS month_start_date,
		r.rental_id,
		r.customer_id,
		i.film_id,
		fc.category_id,
		c.name AS category_name,
		r.rental_date,
		r.return_date,
		p.amount
	FROM
		rental r
	INNER JOIN inventory i ON r.inventory_id = i.inventory_id
	INNER JOIN film f ON i.film_id = f.film_id
	INNER JOIN film_category fc ON f.film_id = fc.film_id
	INNER JOIN category c ON fc.category_id = c.category_id
	INNER JOIN payment p ON r.rental_id = p.rental_id
)
SELECT
	Month_start_date,
	Rental_id,
	Customer_id,
	Film_id,
	Category_id,
	Category_name,
	Rental_date,
	Return_date,
	amount
FROM
	DetailedRentals;
```

E. SQL code to create trigger event

```
CREATE OR REPLACE FUNCTION update_monthly_summary()
RETURNS TRIGGER AS $$
BEGIN
```

```
-- Update total rentals and average spend
UPDATE monthly_summary
SET
	total_rentals = total_rentals + 1,
	average_spend = TO_CHAR(((
		(REPLACE(average_spend, '$', '')::NUMERIC * total_rentals) + NEW.amount) /
	(total_rentals + 1)), 'FM$999,999.00')
WHERE month_start_date = NEW.month_start_date;

-- Update genre counts
WITH GenreCounts AS (
	SELECT
		category_id,
		COUNT(*) AS genre_count
	FROM
		monthly_detailed
	WHERE
		month_start_date = NEW.month_start_date
	GROUP BY
		category_id
)
UPDATE monthly_summary
SET
	top_genre_id = (SELECT category_id FROM GenreCounts ORDER BY genre_count DESC
LIMIT 1),
	top_genre_count = (SELECT genre_count FROM GenreCounts ORDER BY genre_count DESC
LIMIT 1),
	second_top_genre_id = (SELECT category_id FROM GenreCounts ORDER BY genre_count
DESC OFFSET 1 LIMIT 1),
	second_top_genre_count = (SELECT genre_count FROM GenreCounts ORDER BY
genre_count DESC OFFSET 1 LIMIT 1),
	least_rented_genre_id = (SELECT category_id FROM GenreCounts ORDER BY genre_count
ASC LIMIT 1),
	least_rented_genre_count = (SELECT genre_count FROM GenreCounts ORDER BY
genre_count ASC LIMIT 1)
	WHERE month_start_date = NEW.month_start_date;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_update_monthly_summary
AFTER INSERT ON monthly_detailed
FOR EACH ROW
EXECUTE FUNCTION update_monthly_summary();
```

F. Stored procedure in text format

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

-- Clear the contents of the detailed table and summary table
TRUNCATE TABLE monthly_detailed;
TRUNCATE TABLE monthly_summary;

 -- Insert fresh data into the detailed table
INSERT INTO monthly_detailed (month_start_date, rental_id, customer_id, film_id, category_id,
category_name, rental_date, return_date, amount)
WITH DetailedRentals AS (
	SELECT
		format_year_month(DATE_TRUNC('month', r.rental_date)) AS month_start_date,
		r.rental_id,
		r.customer_id,
		i.film_id,
		fc.category_id,
		c.name AS category_name,
		r.rental_date,
		r.return_date,
		p.amount
	FROM
		rental r
	INNER JOIN inventory i ON r.inventory_id = i.inventory_id
	INNER JOIN film f ON i.film_id = f.film_id
	INNER JOIN film_category fc ON f.film_id = fc.film_id
	INNER JOIN category c ON fc.category_id = c.category_id
	INNER JOIN payment p ON r.rental_id = p.rental_id
 )
SELECT
	month_start_date,
	rental_id,
	customer_id,
	film_id,
	category_id,
	category_name,
	rental_date,
	return_date,
	amount
FROM
	DetailedRentals;
```

```
-- Insert fresh data into the summary table
INSERT INTO monthly_summary (month_start_date, total_rentals, average_spend, top_genre_id,
top_genre, top_genre_count, second_top_genre_id, second_top_genre, second_top_genre_count,
least_rented_genre_id, least_rented_genre, least_rented_genre_count)
SELECT
	month_start_date,
	COUNT(DISTINCT rental_id) AS total_rentals,
	TO_CHAR(AVG(amount), 'FM$999,999.00') AS average_spend,
	(SELECT category_id FROM (
		SELECT category_id, COUNT(*) AS count
		FROM monthly_detailed
		WHERE month_start_date = m.month_start_date
		GROUP BY category_id
		ORDER BY count DESC
		LIMIT 1
	) AS t) AS top_genre_id,
	(SELECT name FROM category WHERE category_id = top_genre_id) AS top_genre,
	(SELECT count FROM (
		SELECT category_id, COUNT(*) AS count
		FROM monthly_detailed
		WHERE month_start_date = m.month_start_date
		GROUP BY category_id
		ORDER BY count DESC
		LIMIT 1
	) AS t) AS top_genre_count,
	(SELECT category_id FROM (
		SELECT category_id, COUNT(*) AS count
		FROM monthly_detailed
		WHERE month_start_date = m.month_start_date
		GROUP BY category_id
		ORDER BY count DESC
		OFFSET 1
		LIMIT 1
	) AS t) AS second_top_genre_id,
	(SELECT name FROM category WHERE category_id = second_top_genre_id) AS
second_top_genre,
	(SELECT count FROM (
		SELECT category_id, COUNT(*) AS count
		FROM monthly_detailed
		WHERE month_start_date = m.month_start_date
		GROUP BY category_id
		ORDER BY count DESC
		OFFSET 1
		LIMIT 1
	) AS t) AS second_top_genre_count,
	(SELECT category_id FROM (
		SELECT category_id, COUNT(*) AS count
		FROM monthly_detailed
		WHERE month_start_date = m.month_start_date
```

```
                GROUP BY category_id
                ORDER BY count ASC
                LIMIT 1
        ) AS t) AS least_rented_genre_id,
        (SELECT name FROM category WHERE category_id = least_rented_genre_id) AS
least_rented_genre,
        (SELECT count FROM (
                SELECT category_id, COUNT(*) AS count
                FROM monthly_detailed
                WHERE month_start_date = m.month_start_date
                GROUP BY category_id
                ORDER BY count ASC
                LIMIT 1
        ) AS t) AS least_rented_genre_count
FROM
        monthly_detailed m
GROUP BY
        month_start_date;
END;
$$;
```

1. Job Scheduling Tool

A relevant job scheduling tool that can be used to automate the stored procedure is pg_cron. I prefer refreshing data daily, but a weekly or monthly refresh would also be fine.

Ensure pg_cron is installed and enter the following:

```
SELECT cron.schedule('daily_refresh', '0 0 * * *', 'CALL refresh_report_data()');
```