DVD Rental Database

Corey Lehman

Overview

The DVD Rental Database is a comprehensive system that manages various aspects of a DVD rental business, including inventory, customers, staff, rentals, and payments. In this project, I conducted an ETL (Extract, Transform, Load) process to refine the data and create tables that enable in-depth analysis.

Using tools such as pgAdmin 4 (PostgreSQL) and Power BI (data visualization), I extracted relevant data from the existing database, transformed it into meaningful insights, and loaded it into structured tables for easy querying and reporting. This process allowed me to address four key business questions: identifying the top 10 revenue-generating movies, determining the top 5 revenue-generating movie categories, analyzing employee sales and rentals month-over-month, and evaluating store sales and rentals month-over-month.

Creating the tables

In this project, I created several refined tables to facilitate data analysis. The `film_revenue_table` captures metrics for each film, including title, rental cost, category, number of rentals, and total revenue. The `category_revenue_table` summarizes the number of films, total rentals, total revenue, and the last rental date for each category. The `employee_revenue_table` tracks employee rentals and sales on a month-by-month basis. Finally, the `store_revenue_table` compiles monthly data for each store, including total rentals, revenue, city, country, and postal code. These tables were designed and populated using an ETL process with tools such as pgAdmin 4 and Power BI.

```
/* Code starts here */
	-- Film film revenue table
	CREATE TABLE film_revenue_table (
		film_title  TEXT NOT NULL,
		rental_cost DECIMAL (10,2) NOT NULL,
		category_name TEXT NOT NULL,
		number_of_rentals INT NOT NULL,
		total_revenue DECIMAL(10,2) NOT NULL,
		film_id INT PRIMARY KEY,
		FOREIGN KEY (film_id) REFERENCES film(film_id),
		CONSTRAINT unique_film_id UNIQUE (film_id)
	);
	-- Category revenue table
	CREATE TABLE category_revenue_table (
		category_name TEXT NOT NULL,
		number_of_films INT NOT NULL,
		total_rentals INT NOT NULL,
		total_revenue DECIMAL(10,2) NOT NULL,
		last_rental_date DATE NOT NULL,
		category_id INT PRIMARY KEY,
		FOREIGN KEY (category_id) REFERENCES category(category_id),
		CONSTRAINT unique_category_name UNIQUE (category_name)
	);
	-- Employee revenue table
	CREATE TABLE employee_revenue_table (
		staff_id INT NOT NULL,
		employee_name TEXT NOT NULL,
		employee_email TEXT NOT NULL,
```

```
    employee_rentals INT NOT NULL,
    employee_sales DECIMAL(10,2) NOT NULL,
    month TEXT NOT NULL,
    FOREIGN KEY (staff_id) REFERENCES staff(staff_id)
);
-- Store revenue table
CREATE TABLE store_revenue_table (
    store_id INT NOT NULL,
    store_rentals INT NOT NULL,
    store_revenue DECIMAL(10,2) NOT NULL,
    city TEXT NOT NULL,
    country TEXT NOT NULL,
    post_code TEXT NOT NULL,
    month TEXT NOT NULL,
    FOREIGN KEY (store_id) REFERENCES store(store_id)
);
```

User-Defined Function

I created the user-defined function date_to_month to transform the date of each sale into the corresponding month name. This function converts a timestamp into a textual representation of the month, which assists in charting monthly sales for employees and stores. Using this function, the sales data can be easily aggregated and visualized monthly.

```
/* Code starts here */
    -- Create a function to update DATE to month (e.g. 'February')
    CREATE OR REPLACE FUNCTION date_to_month(dateInput TIMESTAMP)
    RETURNS TEXT AS $$
    BEGIN
        RETURN TO_CHAR(dateInput, 'Month');
    END
    $$
    LANGUAGE plpgsql;
```

Creating a stored procedure to refresh each table

The stored procedure `fetchData()` is designed to refresh the data in each of the refined tables: `film_revenue_table`, `category_revenue_table`, `employee_revenue_table`, and `store_revenue_table`. This procedure truncates the existing data in these tables to ensure they are empty before new data is inserted. It then populates each table with updated data by aggregating information from various source tables. The procedure computes key metrics such as total revenue, number of rentals, and month-to-month performance for films, categories, employees, and stores. This automated refresh process ensures that the data in the refined tables is always current and accurate, supporting ongoing business analysis.

```
/* Code starts here */
    -- Create procedure to refresh data (use a job scheduler to set up routine refresh)
    CREATE OR REPLACE FUNCTION fetchData()
    LANGUAGE plpgsql
    AS $$
    BEGIN
    -- Clear all tables
    TRUNCATE TABLE
    film_revenue_table,category_revenue_table,employee_revenue_table,store_revenue_table;
```

```
-- Populate tables (extract raw data from database)
INSERT INTO film_revenue_table
	SELECT
		f.title,
		f.rental_rate,
		c.name,
		COUNT(r.rental_id),
		SUM(p.amount),
		f.film_id

	FROM film f
	JOIN film_category fc ON f.film_id = fc.film_id
	JOIN category c ON fc.category_id = c.category_id
	JOIN inventory i ON f.film_id = i.film_id
	JOIN rental r ON i.inventory_id = r.inventory_id
	JOIN payment p ON r.rental_id = p.rental_id
GROUP BY
	f.title,
	c.name,
	f.film_id,
	f.rental_rate;
INSERT INTO category_revenue_table
	SELECT
		c.name,
		COUNT(DISTINCT f.title),
		COUNT(r.rental_id),
		SUM(p.amount),
		MAX(r.return_date),
		c.category_id
	FROM category c
	JOIN film_category fc
		ON c.category_id = fc.category_id
	JOIN film f
		ON fc.film_id = f.film_id
	JOIN inventory i
		ON f.film_id = i.film_id
	JOIN rental r
		ON i.inventory_id = r.inventory_id
	JOIN payment p
		ON r.rental_id = p.rental_id
GROUP BY
	c.name,
	c.category_id;

INSERT INTO employee_revenue_table
	SELECT
		s.staff_id,
		CONCAT(s.first_name,' ',s.last_name),
		s.email,
		COUNT(r.rental_ID),
		SUM(p.amount),
		date_to_month(r.rental_date)
	FROM rental r
	JOIN payment p
		ON r.rental_id = p.rental_id
```

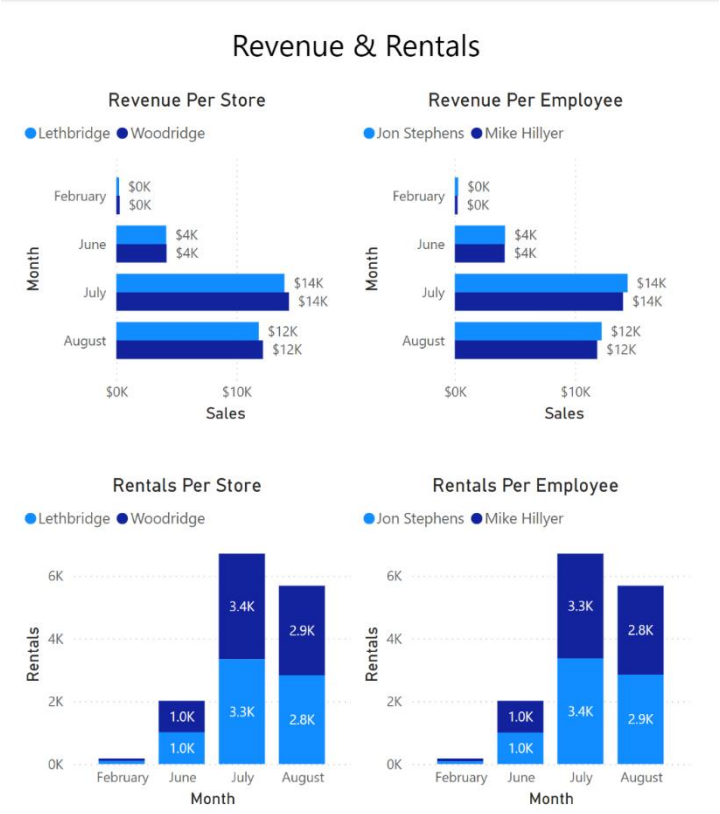
```
        JOIN staff s
                ON p.staff_id = s.staff_id
GROUP BY
        s.staff_id,
        s.first_name,
        s.last_name,
        s.email,
        date_to_month(r.rental_date);

INSERT INTO store_revenue_table
        SELECT
                store.store_id,
                COUNT(r.rental_ID),
                SUM(p.amount),
                city.city,
                country.country,
                a.postal_code,
                date_to_month(r.rental_date)
        FROM country
        JOIN city
                ON country.country_id = city.country_id
        JOIN address a
                ON city.city_id = a.city_id
        JOIN store
                ON a.address_id = store.address_id
        JOIN staff s
                ON store.store_id = s.store_id
        JOIN payment p
                ON s.staff_id = p.staff_id
        JOIN rental r
                ON p.rental_id = r.rental_id
GROUP BY store.store_id,
        city.city,
        country.country,
        a.postal_code,
        date_to_month(r.rental_date);
END;
$$
```

Power BI Model

I imported the server and tables into Power BI using SQL queries to visualize the data. I connected Power BI to the PostgreSQL server, imported the refined tables created in the ETL process, and used SQL queries to pull the necessary data into Power BI. This enabled me to create dynamic visualizations that provide insights into various aspects of the DVD rental business. The visualizations include dashboards that display revenue and rentals per store and employee on a month-by-month basis and the top revenue-generating movies and categories. These dashboards are instrumental in analyzing performance and making data-driven decisions.

Here are screenshots of the visualizations:

- Revenue and Rentals Dashboard:



- Top Movies and Categories Dashboard: