DBMS Lab 4 Demo

All examples here are with respect to the dvdrental database provided to you.

Introduction

This document provides an in-depth explanation of the key SQL concepts used in Week 4's assignment. Topics include **Recursive Queries, Aggregate Functions, Ranking Methods, Joins, and Subqueries**. Each section includes a detailed explanation, examples, and practical applications.

Recursive Queries

Recursive queries allow repeated execution of a query until a defined stopping condition is met, enabling hierarchical data traversal and sequence generation.

Definition and Key Features

- Uses `WITH RECURSIVE` to define the recursive structure.
- Contains two parts: **Base Case** (initial result set) and **Recursive Case** (iterative logic applied to previous results).
- Stops executing when no new rows are produced.

Example: Generating a Sequence from 1 to 10

```
WITH RECURSIVE numbers(n) AS (

        SELECT 1  -- Base Case

        UNION ALL

        SELECT n+1 FROM numbers WHERE n < 10  -- Recursive Case

)

SELECT * FROM numbers;
```

Practical Application: Recursive queries are used for navigating organizational hierarchies, handling bill of materials (BOM) structures, and generating sequence numbers dynamically.

Joins

Joins retrieve data from multiple tables based on relationships between columns.

Types of Joins

- **INNER JOIN**: Returns records with matching values in both tables.
- **LEFT JOIN**: Returns all records from the left table and matching records from the right table.
- **RIGHT JOIN**: Returns all records from the right table and matching records from the left table.
- **FULL OUTER JOIN**: Returns all records when there is a match in either left or right table.

Example: INNER JOIN - Finding Customer Rentals

```
SELECT c.customer_id, c.first_name || ' ' || c.last_name AS customer_name,
COUNT(r.rental_id) AS rental_count

FROM customer c

INNER JOIN rental r ON c.customer_id = r.customer_id

GROUP BY c.customer_id;
```

Aggregate Functions

Aggregate functions compute values across multiple rows, providing insights into large datasets.

Practical Application

Aggregate functions help in summarizing data such as total sales, customer spending, and performance metrics.

Example: Finding the Average Rental Rate per Category

```
SELECT c.name AS category_name, ROUND(AVG(f.rental_rate), 3) AS avg_rental_rate

FROM category c

JOIN film_category fc ON c.category_id = fc.category_id

JOIN film f ON fc.film_id = f.film_id
```

```
GROUP BY c.name;
```

Nested Subqueries

Subqueries allow dynamic filtering or value computation within another SQL query.

Practical Application

Subqueries are useful for conditional filtering, checking existence, and performing calculations within a larger query.

Example: Finding Customers Who Spent More Than the Average Payment

```
SELECT customer_id, first_name, last_name

FROM customer

WHERE customer_id IN (

        SELECT customer_id FROM payment

        GROUP BY customer_id

        HAVING SUM(amount) > (SELECT AVG(amount) FROM payment)

);
```

Window Functions

Window functions perform calculations across a set of table rows while retaining individual row identities.

Definition and Key Features

- Unlike aggregate functions (which collapse multiple rows into a single result), **window functions operate over a specified range of rows using the `OVER()` clause**.
- Useful for **ranking, running totals, moving averages, and comparisons within a partition**.
- The **`PARTITION BY` clause** allows grouping of rows before applying the function.

Example: Ranking Customers by Total Payment

```
SELECT customer_id, first_name || ' ' || last_name AS customer_name,
      SUM(amount) OVER (PARTITION BY customer_id) AS total_payment,
      RANK() OVER (ORDER BY SUM(amount) OVER (PARTITION BY customer_id) DESC) AS
rank FROM payment JOIN customer USING(customer_id);
```

Practical Application

Used for **ranking customers by spending, calculating running totals in financial reports, and analyzing time-series data**.

Common Table Expressions (CTEs)

A Common Table Expression (CTE) allows creating a temporary result set that can be referenced within the main query, making complex queries more readable and modular.

Definition and Key Features

- Defined using the `WITH` keyword, followed by a named temporary result set.
- Can be referenced multiple times within the main query, avoiding redundant subqueries.
- Makes complex hierarchical queries and multi-step calculations more understandable.

Example: Finding Customers with More than 5 Rentals

```
WITH RentalCount AS (

          SELECT customer_id, COUNT(*) AS rental_count

          FROM rental

          GROUP BY customer_id

)

SELECT customer_id FROM RentalCount WHERE rental_count > 5;
```

Practical Application: CTEs are commonly used for hierarchical queries, breaking down large queries into manageable sections, and improving query optimization.

Advanced Query Techniques

This section discusses a few examples of complex SQL queries applied in an inventory management system.

1. Store-Level Analysis: Tracking Customer Rentals Across Multiple Locations

Theory: Store-level analysis helps businesses understand customer engagement across different locations. Identifying customers who rent from multiple stores can offer insights into loyalty trends and regional rental preferences.

Example Query:

```
SELECT c.customer_id, c.first_name || ' ' || c.last_name AS customer_name,
COUNT(DISTINCT i.store_id) AS store_count

FROM customer c

JOIN rental r ON c.customer_id = r.customer_id

JOIN inventory i ON r.inventory_id = i.inventory_id

GROUP BY c.customer_id, c.first_name, c.last_name

HAVING COUNT(DISTINCT i.store_id) > 1;
```

This query identifies customers with a broad rental footprint.

---

2. Top Payment Buckets Analysis: Grouping Payments by Range

Theory: Payment bucket analysis segments transactions into ranges to better understand customer spending behavior. This grouping allows businesses to identify high-value and low-value customer segments.

Example Query:

```
SELECT CASE
    WHEN amount < 5 THEN 'Low Payment'
    WHEN amount BETWEEN 5 AND 15 THEN 'Medium Payment'
    ELSE 'High Payment'
END AS payment_range, COUNT(*) AS frequency
FROM payment
GROUP BY payment_range;
```

This query provides insights into transaction distribution.

---

3. City and Country Sales Insights: Analyzing Rental Trends by Location

Theory: Analyzing rentals at the city and country levels helps businesses track regional performance and identify high-demand areas.

Example Query:

```
SELECT ci.city, co.country, COUNT(r.rental_id) AS total_rentals
FROM city ci
JOIN address a ON ci.city_id = a.city_id
JOIN customer c ON a.address_id = c.address_id
JOIN rental r ON c.customer_id = r.customer_id
JOIN country co ON ci.country_id = co.country_id
GROUP BY ci.city, co.country
ORDER BY total_rentals DESC;
```

This query highlights rental activity by geographic regions.

---

4. Staff Performance Analysis: Identifying High-Performing Staff

Theory: Staff performance analysis involves measuring employee contributions based on key metrics like transaction volume. It helps identify top performers and optimize operational efficiency.

Example Query:

```
SELECT s.first_name || ' ' || s.last_name AS staff_name, st.store_id, COUNT(p.payment_id) AS
total_transactions
FROM staff s
JOIN payment p ON s.staff_id = p.staff_id
JOIN store st ON s.store_id = st.store_id
GROUP BY s.first_name, s.last_name, st.store_id
ORDER BY total_transactions DESC;
```

This query helps in tracking staff performance.

---

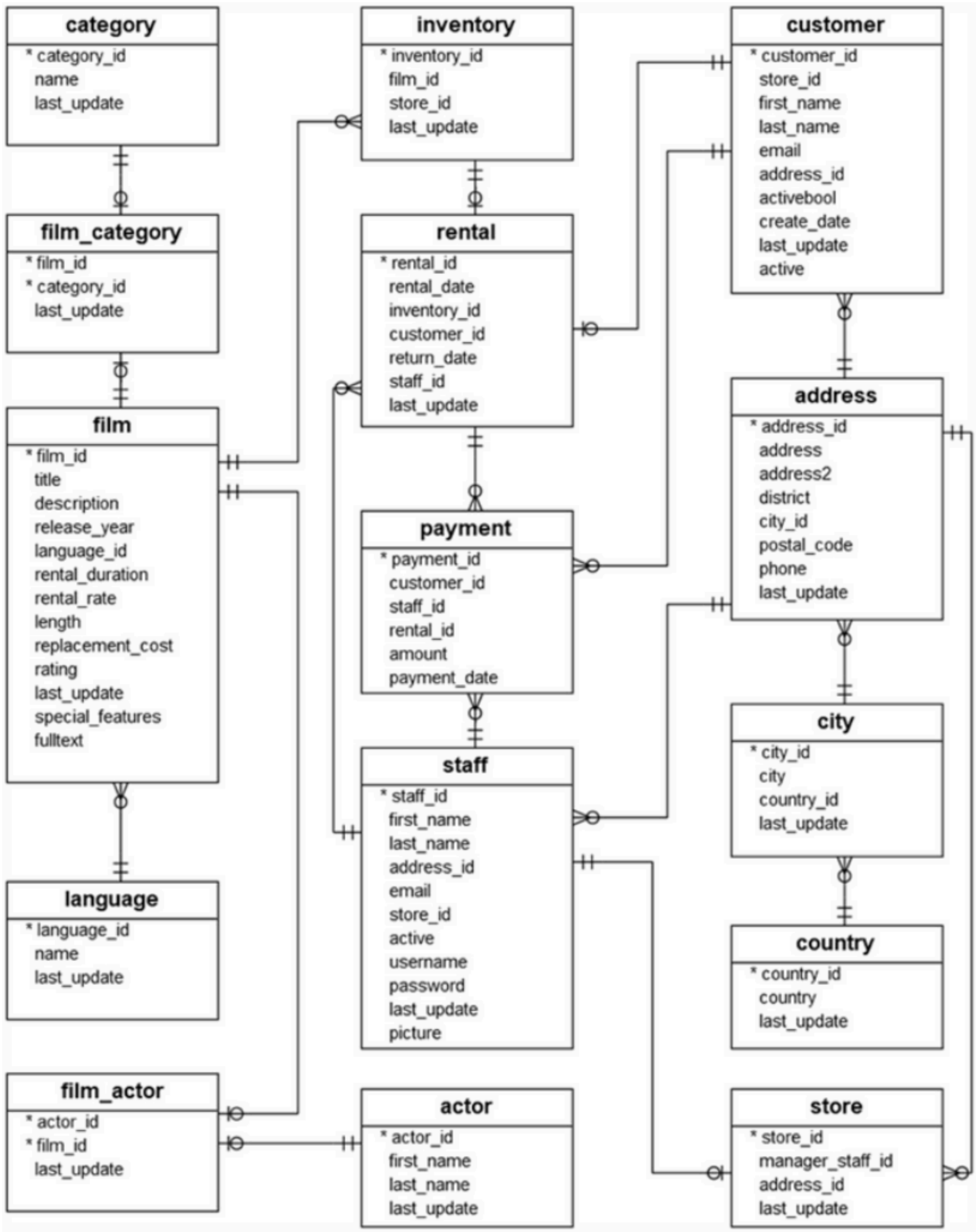
5. Above-Average Sales Analysis: Filtering High-Value Transactions

Theory: Analyzing transactions that exceed the average payment helps identify high-spending customers, which is critical for targeted marketing strategies.

Example Query:

```
WITH AvgPayment AS (
   SELECT AVG(amount) AS avg_payment
   FROM payment
)
SELECT c.first_name || ' ' || c.last_name AS customer_name, SUM(p.amount) AS
total_payment
FROM customer c
JOIN payment p ON c.customer_id = p.customer_id
WHERE p.amount > (SELECT avg_payment FROM AvgPayment)
GROUP BY c.customer_id, c.first_name, c.last_name
ORDER BY total_payment DESC;
```

This query provides insights into customers who spend above the average payment threshold.


category
* category_id
name
last_update
film_category
* film_id
* category_id
last_update
film
* film_id
title
description
release_year
language_id
rental_duration
rental_rate
length
replacement_cost
rating
last_update
special_features
fulltext
language
* language_id
name
last_update
film_actor
* actor_id
* film_id
last_update
inventory
* inventory_id
film_id
store_id
last_update
rental
* rental_id
rental_date
inventory_id
customer_id
return_date
staff_id
last_update
payment
* payment_id
customer_id
staff_id
rental_id
amount
payment_date
staff
* staff_id
first_name
last_name
address_id
email
store_id
active
username
password
last_update
picture
actor
* actor_id
first_name
last_name
last_update
customer
* customer_id
store_id
first_name
last_name
email
address_id
activebool
create_date
last_update
active
address
* address_id
address
address2
district
city_id
postal_code
phone
last_update
city
* city_id
city
country_id
last_update
country
* country_id
country
last_update
store
* store_id
manager_staff_id
address_id
last_update