CoGrammar

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

Basics of SQL Querying

Goals

- Understand the basics of building SQL queries
- Get ready for SQL interview

Sections

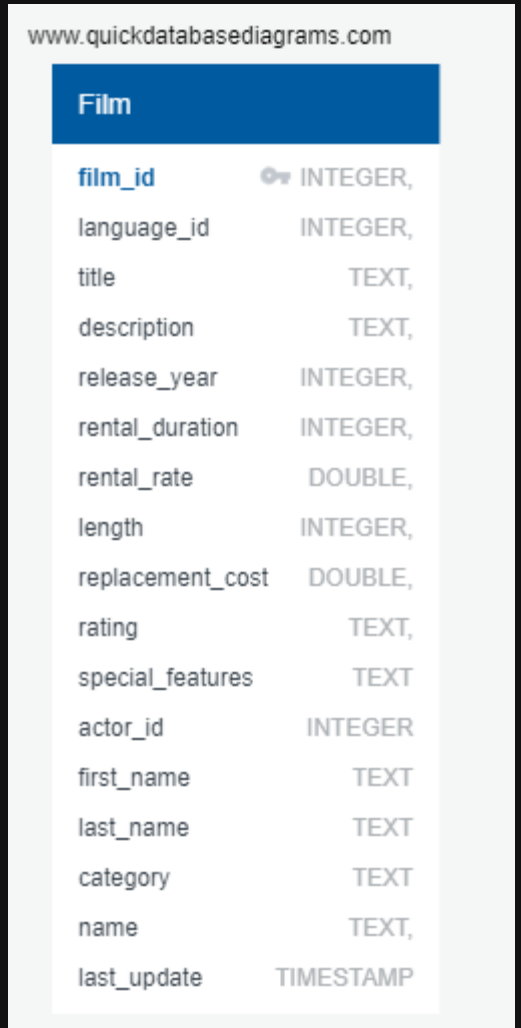
1. Joins
2. Sub-queries
3. Common Table Expressions (CTEs)
4. Views

Order of Execution

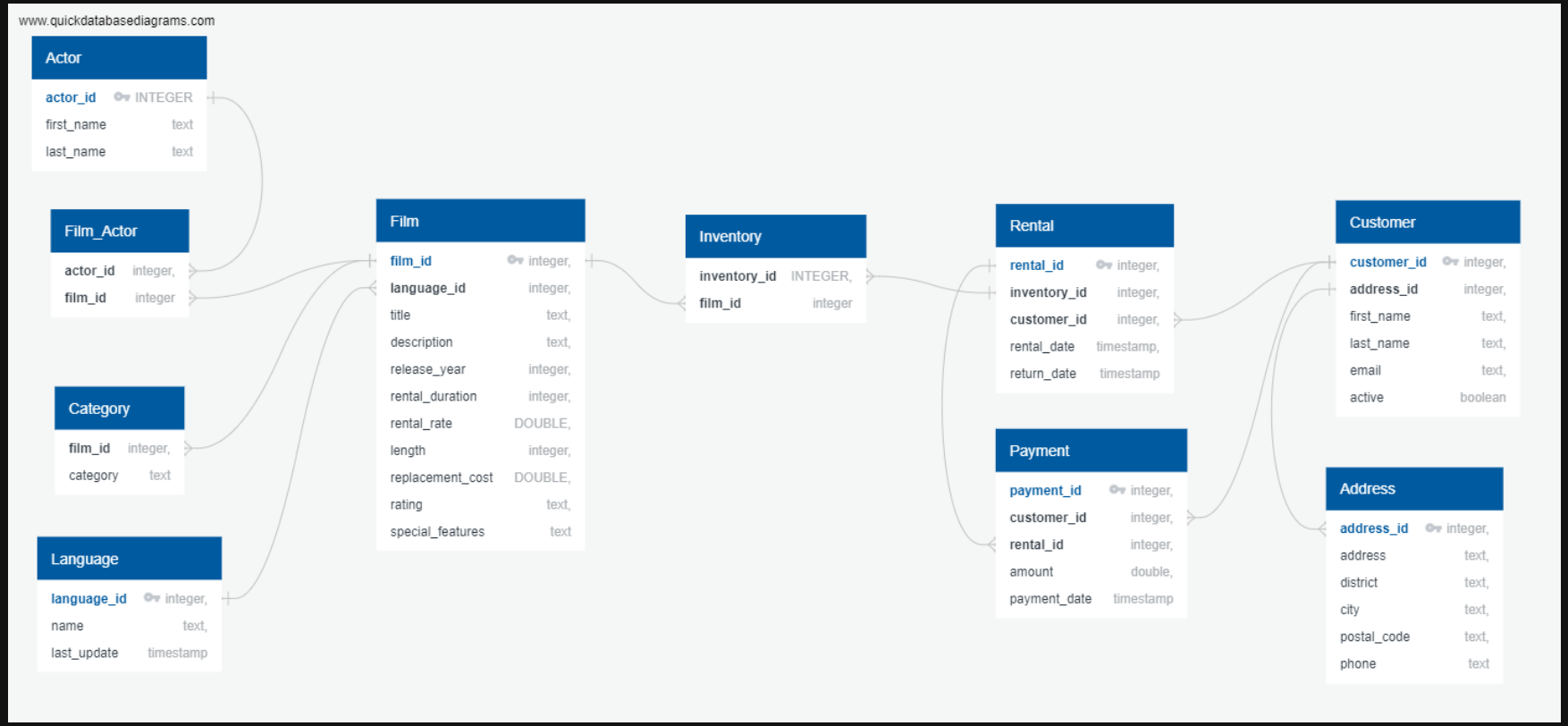
Full Order of Execution

1. FROM / `JOIN`
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

Denormalized table



Normalized Tables



Normalized

Query to get all records

```
SELECT f.*, a.*, c.*,l.*
FROM film f
INNER JOIN film_actor fa ON fa.film_id = f.film_id
INNER JOIN actor a ON fa.actor_id = a.actor_id
INNER JOIN category c ON f.film_id = c.film_id
INNER JOIN language l on f.language_id = l.language_id
```

Unnormalized

Query to get all records

```
SELECT *
FROM film
```

Section 1: CROSS JOIN

```
SELECT f.title as film_title, l.name AS language
FROM film f, language l
```

```
SELECT f.title as film_title, l.name AS language
FROM film f, language l
WHERE f.language_id = l.language_id
```

Notes

- Creates every possible combination for the records in each table
- We can use a `WHERE` clause to make it simulate an `INNER JOIN`
- AVOID USING THIS!!

Section 1: INNER JOIN

```
SELECT f.title, l.name AS language
FROM film AS f
INNER JOIN language l ON f.language_id = l.language_id
```

Notes

- Most common JOIN
- Only joins records with matching values
  - `ON table1.column = table2.column`
- Does not return any null values for missing relationships

Section 1: OUTER JOIN

LEFT JOIN

```
SELECT l.*, f.title
FROM language as l
LEFT JOIN film as f
ON l.language_id = f.language_id
```

Notes

- Left and Right table refers to the order that the tables were called
  - First table (`FROM` clause) will be the left table for example
- `LEFT JOIN` will return all of the records from the left table
- If a record in the left table is missing a relationship in the right table, the missing values will be shown as null
- More commonly used than `RIGHT JOIN`

Section 1: OUTER JOIN

RIGHT JOIN

```
SELECT l.*, f.title
FROM language as l
RIGHT JOIN film as f
ON l.language_id = f.language_id
```

FULL JOIN

```
SELECT l.*, f.title
FROM language as l
FULL JOIN film as f
ON l.language_id = f.language_id
```

Notes

- Left and Right table refers to the order that the tables were called
  - First table (`FROM` clause) will be the left table for example
- Works the exact same way as the `LEFT JOIN` except we will be getting all of the values in the right table and attacting records from the left table
- Not commonly used, a `LEFT JOIN` is usually used by just moving the right table to the left

Section 1: OUTER JOIN

FULL JOIN

```
SELECT l.*, f.title
FROM language as l
FULL JOIN film as f
ON l.language_id = f.language_id
```

Notes

- Takes all of the records from the left and right tables and joins them
- If there are missing relationships in either table, they will be shown as null

Section 1: SELF JOIN

```
SELECT f1.title, f1.length, f1.rental_rate
FROM film f1
INNER JOIN film f2 ON f1.rental_rate = f2.rental_rate
```

Notes

- Used when joining a table to itself
- Useful for looking at relationships within a single table
- Not really effective when the database is normalized

Section 2: Sub-queries

```
SELECT *
FROM rental
WHERE customer_id IN (
    SELECT customer_id
    FROM payment
    WHERE amount > 10
)
```

Notes

- Allows us to perform a query within another query
- Allows us to simplify complex queries

Section 2: Sub-queries

WHERE Clause

```sql
SELECT *
FROM rental
WHERE customer_id IN (
    SELECT customer_id
    FROM payment
    WHERE amount > 10
)
```

```sql
SELECT title, length
FROM film
WHERE length > (
    SELECT AVG(length)
    FROM film
);
```

Notes

- The comparison needs to match the output of the select
- You can think of the select as returning a single value or a list of values

Section 2: Sub-queries

SELECT Clause

```
SELECT title, (
    SELECT COUNT(*)
    FROM inventory AS i
    WHERE f.film_id = i.film_id
) AS total_inventory
FROM film AS f
ORDER BY title
```

Notes

- Can be used to perform aggregation
- Typically used to get single values related to the values being displayed

Section 2: Sub-queries

FROM Clause

```
SELECT full_name, email, address
FROM (
    SELECT CONCAT(c.first_name, ' ', c.last_name)
        AS full_name, c.email, a.address
    FROM customer AS c
    INNER JOIN address AS a
    USING(address_id)
) AS customer_details
```

Notes

- Can be thought of as a custom table being created
- Useful when you want to perform multiple joins

Section 3: Common Table Expressions

```sql
SELECT full_name, email, address
FROM (
    SELECT CONCAT(c.first_name, ' ', c.last_name)
        AS full_name, c.email, a.address
    FROM customer AS c
    INNER JOIN address AS a
    ON a.address_id = c.address_id
) AS customer_details
```

```sql
WITH customer_details (full_name, email, address)
AS (
    SELECT CONCAT(c.first_name, ' ', c.last_name)
        AS full_name, c.email, a.address
    FROM customer AS c
    INNER JOIN address AS a
    ON a.address_id = c.address_id
)

SELECT full_name, email, address
FROM customer_details
```

- Creates a temporary table that can be called in the `SELECT` statement
- Can only be used once

Section 4: Views

```sql
SELECT full_name, email, address
FROM (
    SELECT CONCAT(c.first_name, ' ', c.last_name)
        AS full_name, c.email, a.address
    FROM customer AS c
    INNER JOIN address AS a
    ON a.address_id = c.address_id
) AS customer_details
```

```sql
CREATE VIEW customer_details AS
SELECT CONCAT(c.first_name, ' ', c.last_name)
    AS full_name, c.email, a.address
FROM customer AS c
INNER JOIN address AS a
ON a.address_id = c.address_id;

SELECT full_name, email, address
FROM customer_details
```

- Like functions in programming, they can be created and used when needed
- Can be used to create a pseudo-table for a common query
- Run the query everytime the view is called

Tools: Software

Docker - Set Up Guide

Docker - Download

Azure Data Studio - Download

Tools: Database Setup

Docker Hub - Postgres

Pull Image

```
docker pull postgres
```

Start Database

```
docker run --name some-postgres -p 5432:5432 -e POSTGRES_PASSWORD=mysecretpassword -d postgres
```

Tools: Azure Data Studio

Install PostgreSQL extension

1. Navigate to extensions `(ctrl/⌘ + shift + X)`
2. Search for `PostgresSQL` extension by `Microsoft`
3. Click install

Connecting to database

1. Navigate to 'Connections' `(ctrl/⌘ + shift + D)`
2. In the top bar where you see 'SERVERS' click on the first icon on the left called 'New Connection'
3. Set the following values in the connection tab, leave the rest of the values as they are
   - `Connection type` - PostgreSQL
   - `Server name` - localhost
   - `Suthentication Type` - Password
   - `User name` - postgres
   - `Password` - mysecretpassword (the password you set when running the docker command)
4. Click Connect

TEST YOUR MIGHT

LEETCODE SQL CHALLENGES