CoGrammar

Workshop 8 - Persistent Storage

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: **www.hyperiondev.com/support**
- Report a **safeguarding** incident: **www.hyperiondev.com/safeguardreporting**
- We would love your **feedback** on lectures: **Feedback on Workshops**

Skills Bootcamp Certification Overview

✅ Criteria 4: Original Deadline for Demonstrating Employability without Co-certification

Record a job outcome by 23 September 2024

✅ Criterion 4: Updated Deadline for Imperial College London & University of Manchester Co-certification

Record a job offer by 15 July 2024

Lecture Objectives

- Understand the importance of persistent data storage
- Understand the importance of SQL
- Understand SQL Data Query Language (DQL)
- Understand how to work with JOINs

Background

Social media is a key part of modern life, these systems allow us to connect with people from all over the world.

A social media site provides a good case study based on the wide range of data that is required for user interactions and business purposes.

What is Persistent Data

What is it

- Data that lives beyond the life cycle of the application
- Allows the system to retain states
- Allows for data to be shared between different instances of an application and on different systems.

Dealing With Persistent Data

Types of Data

- Master Data
  - Data from different domains within a business that is used as a central point of truth
- Transactional Data
  - Fast moving data that is required for performing some business function
- Analytical Data
  - Historical data that is used for making business decisions
- Freeform data
  - Artifacts like media, emails, documents

Dealing With Persistent Data

Data Formats

- Structured
  - Data that has strict rules imposed on it when it's being stored
  - Data has to follow certain rules
- Semi-Structured
  - The storage solution doesn't enforce any rules
  - Any rules on the format must be followed by the people working with the data
- Unstructured
  - Data that can not follow a strict set of rules

Data Storage Solutions

Master data
- Structured data
- Usually stored in tabular databases

Transactional
- Can be structured or semi-structured
- Can use tabular, document based, graph etc.

Analytical
- Structured data
- Uses columnar databases

Freeform
- Unstructured
- Uses BLOB storage

Database Options

Structured

- Relational Database Management System (RDBMS)
  - Microsoft SQL Server
  - Oracle SQL
  - PostgreSQL
- Data Warehouse
  - Snowflake
  - Data Bricks

Database Options

Semi-structured

- Not Only SQL (NoSQL)
  - MongoDB
  - Firebase Realtime Database / Firestore
  - Neo4j

Unstructured

- Binary Large Object
  - Microsoft Blob Storage
  - Amazon S3
  - Google Cloud Storage

SQL

Why Learn SQL

- Most popular database language
- Not only used within database engines, but also for analysis and dashboarding tools
- Can be used by many different people within an organisation
  - Software engineers
  - Data analysts
  - Business analysts

Please take a look at the handbook for more background into SQL

SQL Querying

SQL performs a lot of operations, but the main operation that most non admin users will make use of are querying. Having a strong understanding of SQL querying is really important.

SELECT Statement

What is it

- Used to retrieve data from a single or multiple tables
- Falls under the Data Querying Language category of SQL
- Allows us to add search filters so that we get the right results.

```
SELECT title, rental_rate, replacement_cost
FROM film
WHERE release_year BETWEEN 2004 AND 2010
ORDER BY rental_year DESC, replacement_cost
```

SELECT Clause

- Specifies the columns that we would like to get data from
- We can use the `*` to return all columns
- We have to specify 1 or more columns to read form

```
SELECT *
FROM film
```

```
SELECT title, release_year
FROM film
```

FROM Clause

- Used to state the table/s we would like to get data from
- Requires 1 or more tables to be selected

```
SELECT title, release_year
FROM film
```

```
SELECT *
FROM film, category
```

Where Clause

- Filters the results of our search
- We can pass logical and comparison operators
- Comparison Operators
  - < - Less than
  - > - Greater than
  - = - Equal to
  - <> - Equality

```
SELECT title, release_year
FROM film
WHERE release_year = 2010
```

Logical Operators

- We can use logical operators to check different things in our WHERE clause
  - AND - Used like a WHERE if you want to check more than one condition.
  - OR - Checks if one of two conditions is true
  - BETWEEN - Checks if a value falls between a given range
  - IN - Checks if a value is present in a list of values.
  - NOT - Checks the opposite for a condition.

Logical Operators

```
...
WHERE release_year > 2005 AND rental_rate < 3
```

```
...
WHERE release_year <> 2005 OR release_year = 2010
```

```
...
WHERE release_year BETWEEN 2005 AND 2010
```

```
...
WHERE release_year IN (2005, 2008, 2010)
```

```
...
WHERE release_year NOT IN (2005, 2008, 2010)
```

String Comparison

- LIKE
  - Checks if a value exists in a list
  - Requires one a '%' or '_' to determine what is being looked for
    - % - Any value and the specified characters
      - %hello - looks for values ending in hello
      - hello% - looks for values starting with hello
    - _ - There can be any character at the specified position
      - _im - Any letter + im, eg. Jim, Kim, Him
      - __m - eg. Gym, sum, hum

Order By Clause

- Determines the column that should be used for ordering the result
- If more than one column is selected, the one that is stated first will be the main order, the other will be a tie breaker
- We can use ASC and DESC to determine the order, orders in ASC by default

```
SELECT release_year, rental_rate
FROM film
ORDER BY release_year
```

```
SELECT release_year, rental_rate
FROM film
ORDER BY release_year DESC
```

```
SELECT release_year, rental_rate
FROM film
ORDER BY release_year DESC, rental_rate
```

Putting Everything Together

- SELECT
  - Chooses one or more columns
- FROM
  - Determines the table that we are working with
- WHERE
  - Filters the results we are working with
- ORDER BY
  - Orders our output

Order of Execution

How we write the statement

- SELECT
- FROM
- WHERE
- ORDER BY

How the statement is executed

- FROM
- WHERE
- SELECT
- ORDER BY

Importance of the order

When you are working with more advanced queries, you might try to access something that you defined in your select and it won't work, this is because the SELECT wouldn't have been executed yet.

Aliasing

- Allows us to give different names to the values we are working with.
- We can alias tables and columns
- We use the AS keyword for aliasing
- Keep the order of execution in mind because we won't be able to use the column aliases for a lot of operations

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

```
SELECT f.title AS 'film title', l.name as film_language
FROM film AS f, language AS l
WHERE f.language_id = l.language_id
AND l.name = 'English'
ORDER BY film_language
```

Aggregation

- Used to summarise data
- We aggregate against a single column
- Methods
  - COUNT()
  - MIN()
  - MAX()
  - SUM()
  - AVG()

```sql
-- Get total number of records
SELECT COUNT(*) AS total_films
FROM film

-- Sum of all rental_rates
SELECT SUM(rental_rate) AS total_rental_rate
FROM film
```

GROUP BY Clause

- Required when we use an aggregations function
  - Except SUM() and COUNT()
- Values will be aggregated against the column we group by
- We can group by 1 or more columns

```sql
SELECT release_year, SUM(rental_rate) AS total_rental_rate
FROM film
GROUP BY release_year
ORDER BY total_rental_rate
```

```sql
-- For 2005
SELECT SUM(rental_rate) AS total_rental_rate_2005
FROM film
WHERE release_year = 2005

-- For 2006
SELECT SUM(rental_rate) AS total_rental_rate_2005
FROM film
WHERE release_year = 2006
```

Final Order of Execution

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

HAVING Clause

- Used to filter values from our aggregation
- Acts like a WHERE clause for aggregation operations
- We need to redo the aggregation that's in the SELECT if we are using that for our comparison.

```
SELECT release_year AS year, SUM(rental_rate) AS total_rental_rate
FROM film
GROUP BY release_year
HAVING SUM(rental_rate) < 50 -- Comes before SELECT in the order of execution
ORDER BY year
```

JOINS

JOINS

What are JOINs

- When working with relational databases, we like to break our data up into different tables to make our data more consistent.
- Storing data in different tables is good for storage and consistency, but when viewing data, a single table doesn't always have all of the information that we need
- JOINs allow us to combine data from multiple tables into a single output.

CROSS JOIN

- This is an implicit JOIN
- Performed by selecting more than one table in the FROM clause and using the WHERE clause to choose the common fields
- Should be avoided

INNER JOIN

- Most common JOIN
- JOINs two tables based on shared properties
  - Finds the same value in the selected columns and links them between the tables
- Doesn't include null values in the output.

id	name
1	tim
2	jim
3	kim
4	joe

id	age
3	24
4	25
6	26
6	27

id	name	age
3	kim	24
4	joe	25

OUTER JOIN

- We join the left and right tables
- LEFT table is the first table that is called
- RIGHT table will be the second table that is called.
- Returns all values from a specific table, if there are no matching records, null values will be returned
- We can do a LEFT, RIGHT and FULL Join
  - LEFT - Joins by the left table
  - RIGHT - Joins by the right table
  - FULL - Joins all data from both tables

LEFT JOIN

Left Table

id	name
1	tim
2	jim
3	kim
4	joe

Right table

id	age
3	24
4	25
6	26
6	27

id	name	age
1	tim	null
2	jim	null
3	kim	24
4	joe	25

RIGHT JOIN

Left Table

id	name
1	tim
2	jim
3	kim
4	joe

Right table

id	age
3	24
4	25
6	26
7	27

id	name	age
3	kim	24
4	joe	25
6	null	26
7	null	27

Outer JOIN

Left Table

id	name
1	tim
2	jim
3	kim
4	joe

Right table

id	age
3	24
4	25
6	26
7	27

id	name	age
1	tim	null
2	jim	null
3	kim	24
4	joe	25
6	null	26
7	null	27

CoGrammar

Q & A SECTION

Please use this time to ask any questions relating to the topic, should you have any.

CoGrammar

Thank you for joining!