Relational Databases

Aggregation and Grouping

8 February 2024

克明峻德，格物致知

Quick Recap: Joining Tables

Key Concepts:

- Entity and Referential Integrity ensure data consistency in business through Primary Keys for uniqueness within tables and Foreign Keys for accurate cross-table references.
- For-Each Semantics offers a conceptual perspective of SQL queries, envisioning them as row-by-row evaluations, aiding in query formulation and understanding their logical processing within the database.

SQL Query Example

```
SELECT Title, Author
FROM Books
WHERE Genre = ’Education’;
```

- **For-Each Semantics**

```
for each row in Books:
    if (row.Genre == 'Eduction'):
        output row.Title
```

- **Purpose:** Retrieves titles from the 'Books' table where the genre is 'Education'.
- **Output Table:**

```
| Title                    | Author  |
| ------------------------ | ------- |
| The History of Databases | D. Base |
| ...                      | ...     |
```

让未来生长
5G
联通世界 创享美好智慧生活
Connect the world to innovate and share a good smart living

A mobile carrier finds it challenging to create effective marketing campaigns due to diverse customer usage behaviors.

The company recognizes the need for a methodical approach for more personalized marketing.

Q: How can the company develop more targeted marketing campaigns?

A: The company can segment customers by aggregating customer data into distinct groups based on their usage patterns.

Actionable Results

Summarizing data is crucial in decision-making and conveying information succinctly.

- Which genre is most popular among our members?
  - Use COUNT to determine the number of times books from each genre are borrowed.
- How active are our library members?
  - Use SUM to calculate the total number of books borrowed by each member.
- What's the average rating of books in a particular genre?
  - Use AVG to find the average rating of books across different genres.
- Which book has been borrowed the most?
  - Use MAX to identify the book that has been borrowed the most times.
- What is the least popular book in our collection?
  - Use MIN to find the book with the least number of borrowings.

These aggregations help in understanding user preferences, managing inventory, and enhancing overall library services.

Key Considerations

Example Query

- SQL Query:

```sql
SELECT AVG(R.Rating)
FROM BookRatings;
```

- **Purpose:** To find the average book rating.

Challenges:

- **Data Duplication:** Joins can duplicate rows, affecting aggregate results.
- **Correct Contex:** Ensuring aggregation aligns with the intended query goal, e.g., average per genre.

Insights:

- **Data Duplication:** Aggregation must account for the data structure and relationships to yield accurate insights.

Aggregation Semantics

Exploring Aggregation

- Aggregations are crucial for summarizing and understanding large datasets, as they can provide insights like averages, totals, maximums, and minimums.

Query Example

- **SQL Query:**

```sql
SELECT AVG(Rating) AS AverageRating
FROM BookRatings;
```

- **Propose:** Computes the average of all values in the "Rating" column of the "BookRatings" table.
- **Output Table:**

```
| AverageRating |
|---------------|
| 3.9           |
```

Understanding GROUP BY

Using GROUP BY for Detailed Aggregation

- The GROUP BY clause in SQL is a powerful tool for dividing data into groups that share common attributes. It allows for more granular analysis by aggregating data within each group separately.

Query Example

- **SQL Query:**

```sql
SELECT b.Genre, AVG(br.Rating) AS AverageRating
FROM Books b
INNER JOIN BookRatings br ON b.BookID = br.BookID
GROUP BY b.Genre;
```

- **Propose:** Computes the average book ratings by genre.
- **Output Table:**

```
| Genre      | AverageRating |
|------------|---------------|
| Technology | 4.2           |
| Science    | 3.8           |
| Fiction    | 4.0           |
| ...        | ...           |
```

Relational Algebra Equivalencies

Example Translation

- SQL Query:

```
SELECT b.Genre, AVG(br.Rating) AS AverageRating
FROM Books b
INNER JOIN BookRatings br ON b.BookID = br.BookID
GROUP BY b.Genre;
```

- Relational Algebra:

```
γGenre; AVG(Rating)→AverageRating (ΠGenre, Rating (Books ⋈ BookRatings))
```

```
[Books Table] + [BookRatings Table]
                  |
                  | (⋈ On BookID)
                  V
    [Joined Tables - Books and BookRatings]
                  |
                  | (Π Genre, Rating)
                  V
       [Projected Columns - Genre and Rating]
                  |
                  | (γ Genre; AVG(Rating)→AverageRating)
                  V
          [Result - Average Rating per Genre]
```

Steps Explained:

1. **Inner Join (⋈):** Joins \`Books\` and \`BookRatings\` on the \`BookID\` attribute.
2. **Projection (Π):** Focuses on the relevant columns, \`Genre\` and \`Rating\`.
3. **Grouping (γ):** Applies the grouping operation on the \`Genre\` attribute and calculates the average rating for each genre.

Aggregated Filtering with HAVING

Using HAVING for Conditional Aggregation

- The HAVING clause in SQL filters groups formed by the GROUP BY clause, based on aggregate functions. It's akin to WHERE, but for aggregated group data rather than individual rows.

Query Example

- SQL Query:

```sql
SELECT b.Genre, AVG(br.Rating) AS AverageRating
FROM Books b
INNER JOIN BookRatings br ON b.BookID = br.BookID
GROUP BY b.Genre
HAVING AVG(br.Rating) > 3.5;
```

- **Propose:** To identify genres that are highly rated by readers (above a certain threshold).

- Output Table:

```
| Genre      | AverageRating |
|------------|---------------|
| Technology | 4.2           |
| Science    | 3.8           |
| Fiction    | 4.0           |
| ...        | ...           |
```

Execution Order: FWGHOS

- SQL Query:

```
SELECT ...
FROM ...
WHERE ...
GROUP BY ...
HAVING ...
ORDER BY ...
```

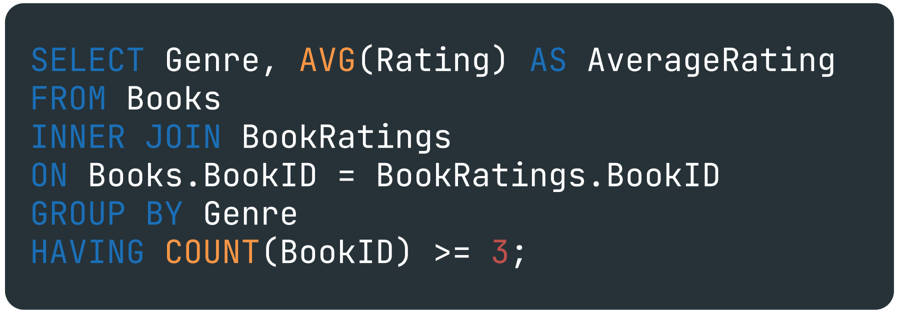
- FWGHOS:

```
[FROM Clause]
      |
      | (Identifies tables, performs joins)
      V
[WHERE Clause]
      |
      | (Filters rows based on conditions)
      V
[GROUP BY Clause]
      |
      | (Groups rows with common values in specified columns)
      V
[HAVING Clause]
      |
      | (Filters groups based on aggregate conditions)
      V
[ORDER BY Clause]
      |
      | (Sorts the result set)
      V
[SELECT Clause]
      |
      | (Selects and displays final columns, applies aggregations)
      V
[Final Result Set]
```

Underestanding Execution Order

FWGHOS Walk Through

- SQL Query:



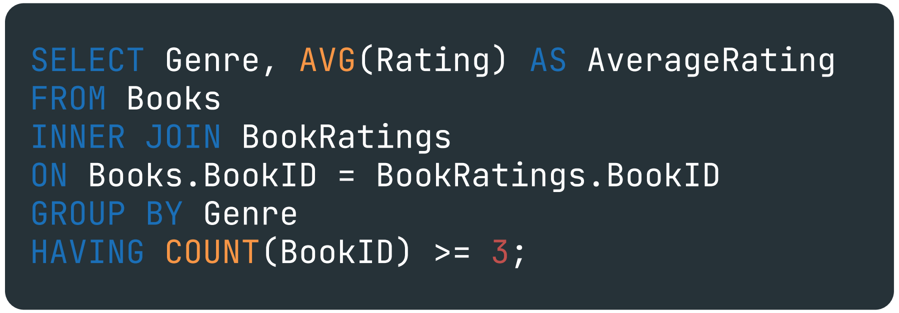
```sql
SELECT Genre, AVG(Rating) AS AverageRating
FROM Books
INNER JOIN BookRatings
ON Books.BookID = BookRatings.BookID
GROUP BY Genre
HAVING COUNT(BookID) >= 3;
```

- **Propose:** To identify genres that are highly rated by readers (above a certain threshold).

Underestanding Execution Order

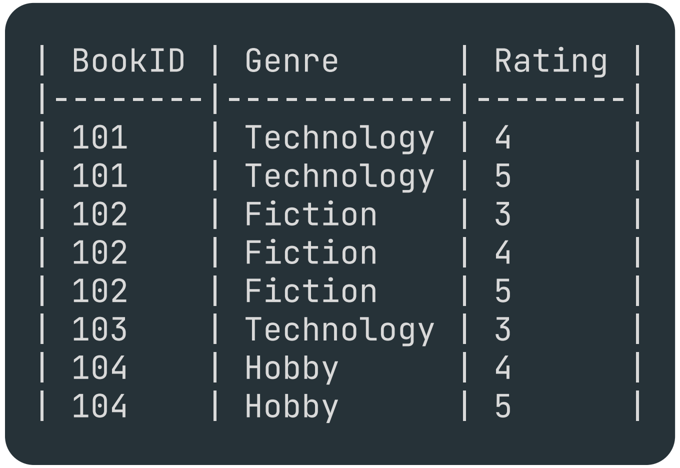
FWGHOS Walk Through

- SQL Query:



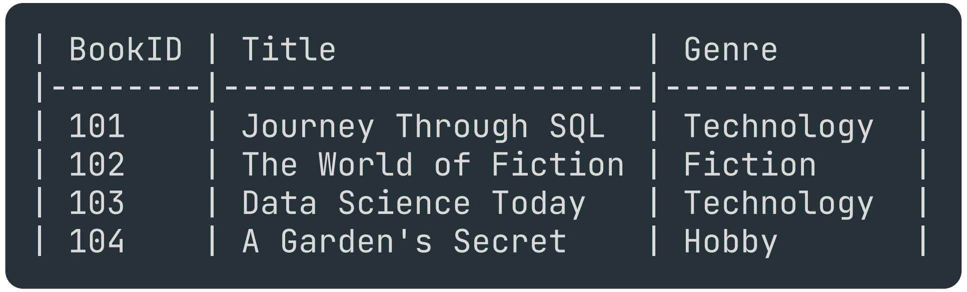
```
SELECT Genre, AVG(Rating) AS AverageRating
FROM Books
INNER JOIN BookRatings
ON Books.BookID = BookRatings.BookID
GROUP BY Genre
HAVING COUNT(BookID) >= 3;
```

- **Propose:** To identify genres that are highly rated by readers (above a certain threshold).

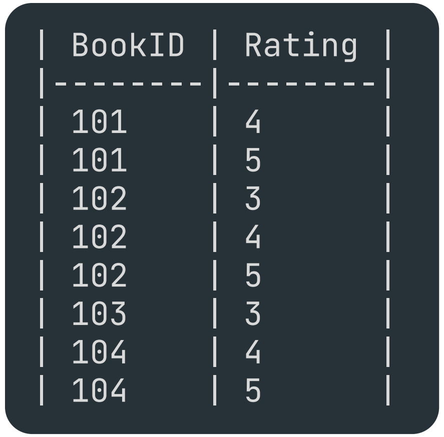


```
| BookID | Genre      | Rating |
|--------|------------|--------|
| 101    | Technology | 4      |
| 101    | Technology | 5      |
| 102    | Fiction    | 3      |
| 102    | Fiction    | 4      |
| 102    | Fiction    | 5      |
| 103    | Technology | 3      |
| 104    | Hobby      | 4      |
| 104    | Hobby      | 5      |
```

[FROM Clause – Books, BookRatings]



```
| BookID | Title                  | Genre        |
|--------|------------------------|--------------|
| 101    | Journey Through SQL    | Technology   |
| 102    | The World of Fiction   | Fiction      |
| 103    | Data Science Today     | Technology   |
| 104    | A Garden's Secret      | Hobby        |
```

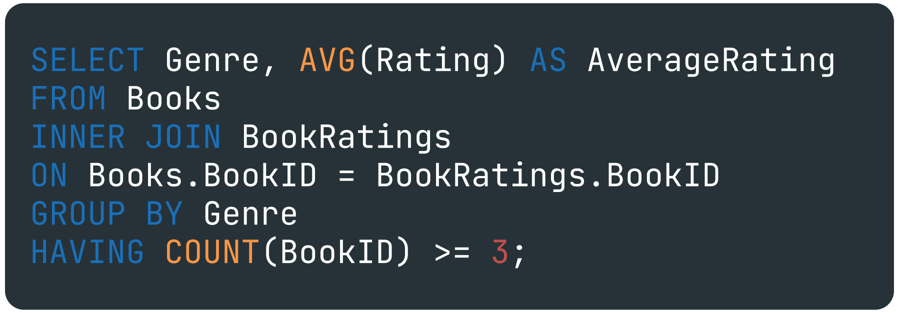


```
| BookID | Rating |
|--------|--------|
| 101    | 4      |
| 101    | 5      |
| 102    | 3      |
| 102    | 4      |
| 102    | 5      |
| 103    | 3      |
| 104    | 4      |
| 104    | 5      |
```

Underestanding Execution Order

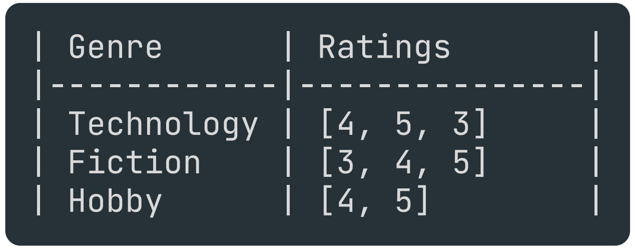
FWGHOS Walk Through

- SQL Query:



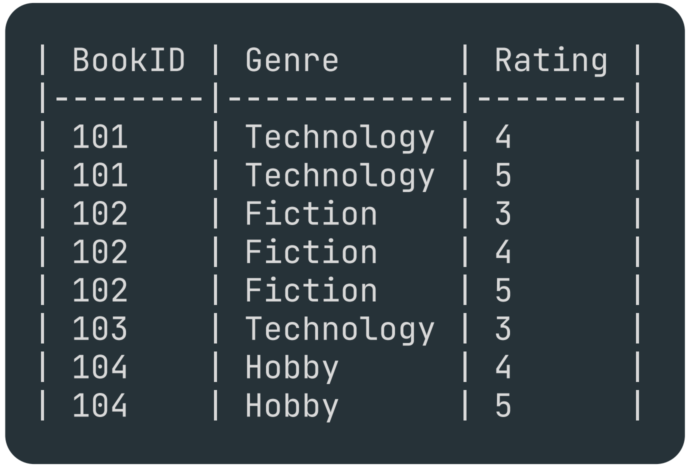
```
SELECT Genre, AVG(Rating) AS AverageRating
FROM Books
INNER JOIN BookRatings
ON Books.BookID = BookRatings.BookID
GROUP BY Genre
HAVING COUNT(BookID) >= 3;
```

- **Propose:** To identify genres that are highly rated by readers (above a certain threshold).



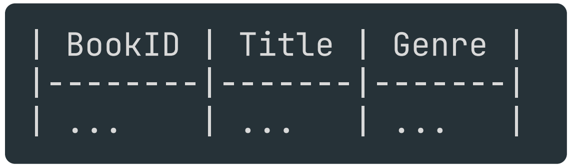
```
| Genre      | Ratings       |
|------------|---------------|
| Technology | [4, 5, 3]     |
| Fiction    | [3, 4, 5]     |
| Hobby      | [4, 5]        |
```

[GROUP BY Clause - Genre]

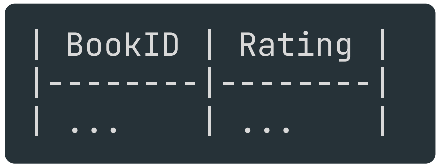


```
| BookID | Genre      | Rating |
|--------|------------|--------|
| 101    | Technology | 4      |
| 101    | Technology | 5      |
| 102    | Fiction    | 3      |
| 102    | Fiction    | 4      |
| 102    | Fiction    | 5      |
| 103    | Technology | 3      |
| 104    | Hobby      | 4      |
| 104    | Hobby      | 5      |
```

[FROM Clause - Books, BookRatings]



```
| BookID | Title | Genre |
|--------|-------|-------|
| ...    | ...   | ...   |
```

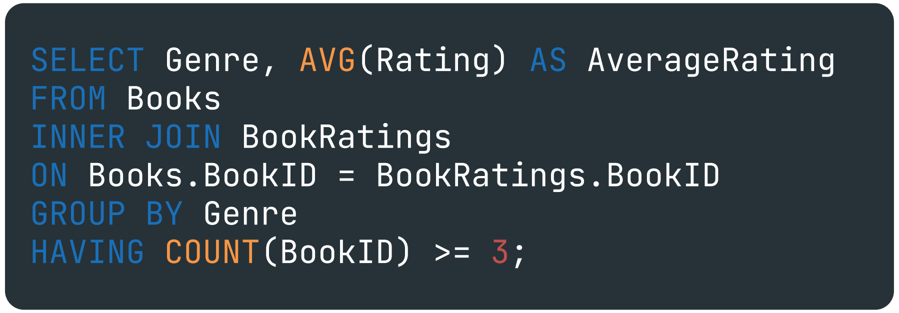


```
| BookID | Rating |
|--------|--------|
| ...    | ...    |
```

Underestanding Execution Order

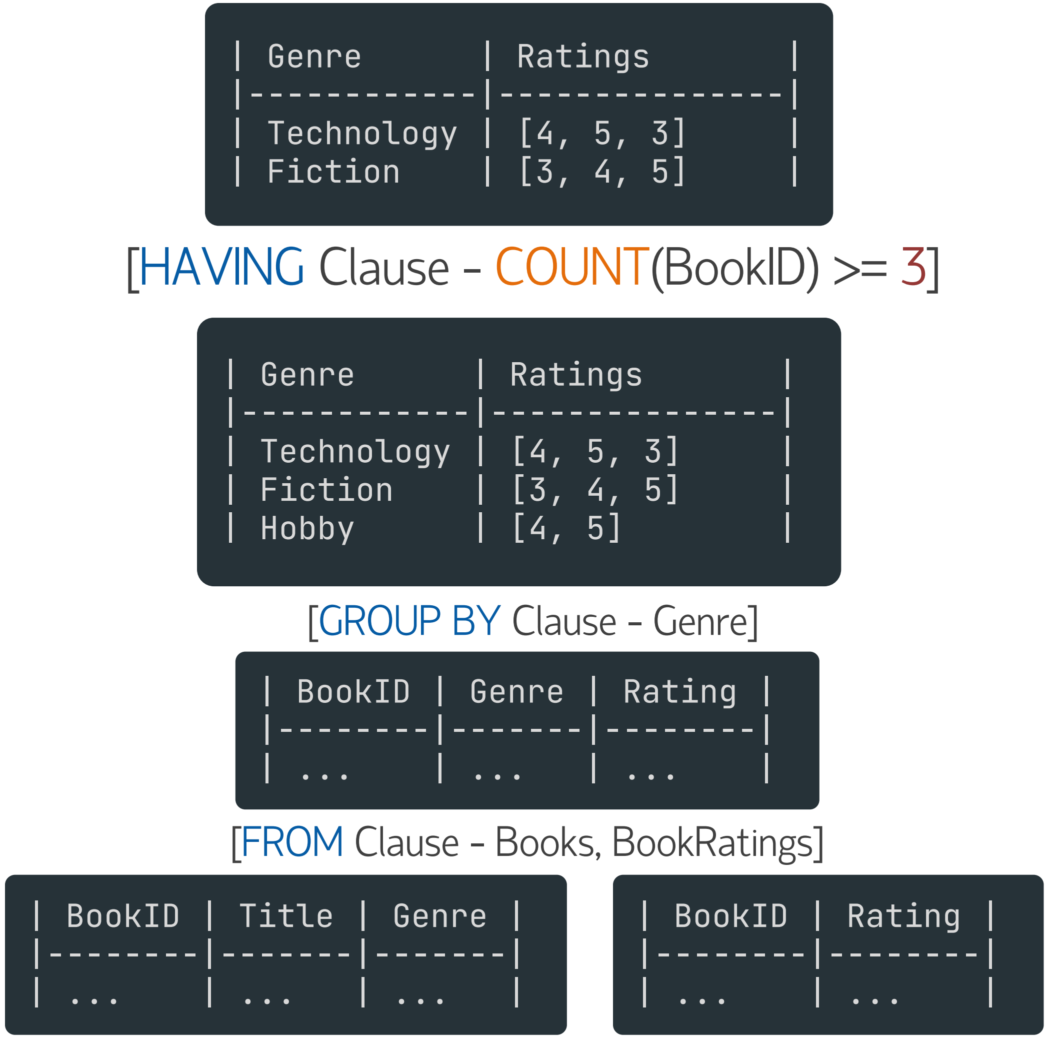
FWGHOS Walk Through

- SQL Query:



```sql
SELECT Genre, AVG(Rating) AS AverageRating
FROM Books
INNER JOIN BookRatings
ON Books.BookID = BookRatings.BookID
GROUP BY Genre
HAVING COUNT(BookID) >= 3;
```

- **Propose:** To identify genres that are highly rated by readers (above a certain threshold).



```
| Genre       | Ratings        |
|-------------|----------------|
| Technology  | [4, 5, 3]      |
| Fiction     | [3, 4, 5]      |
```

[HAVING Clause - COUNT(BookID) >= 3]

```
| Genre       | Ratings        |
|-------------|----------------|
| Technology  | [4, 5, 3]      |
| Fiction     | [3, 4, 5]      |
| Hobby       | [4, 5]         |
```

[GROUP BY Clause - Genre]

```
| BookID | Genre | Rating |
|--------|-------|--------|
| ...    | ...   | ...    |
```

[FROM Clause - Books, BookRatings]

```
| BookID | Title | Genre |
|--------|-------|-------|
| ...    | ...   | ...   |
```

```
| BookID | Rating |
|--------|--------|
| ...    | ...    |
```

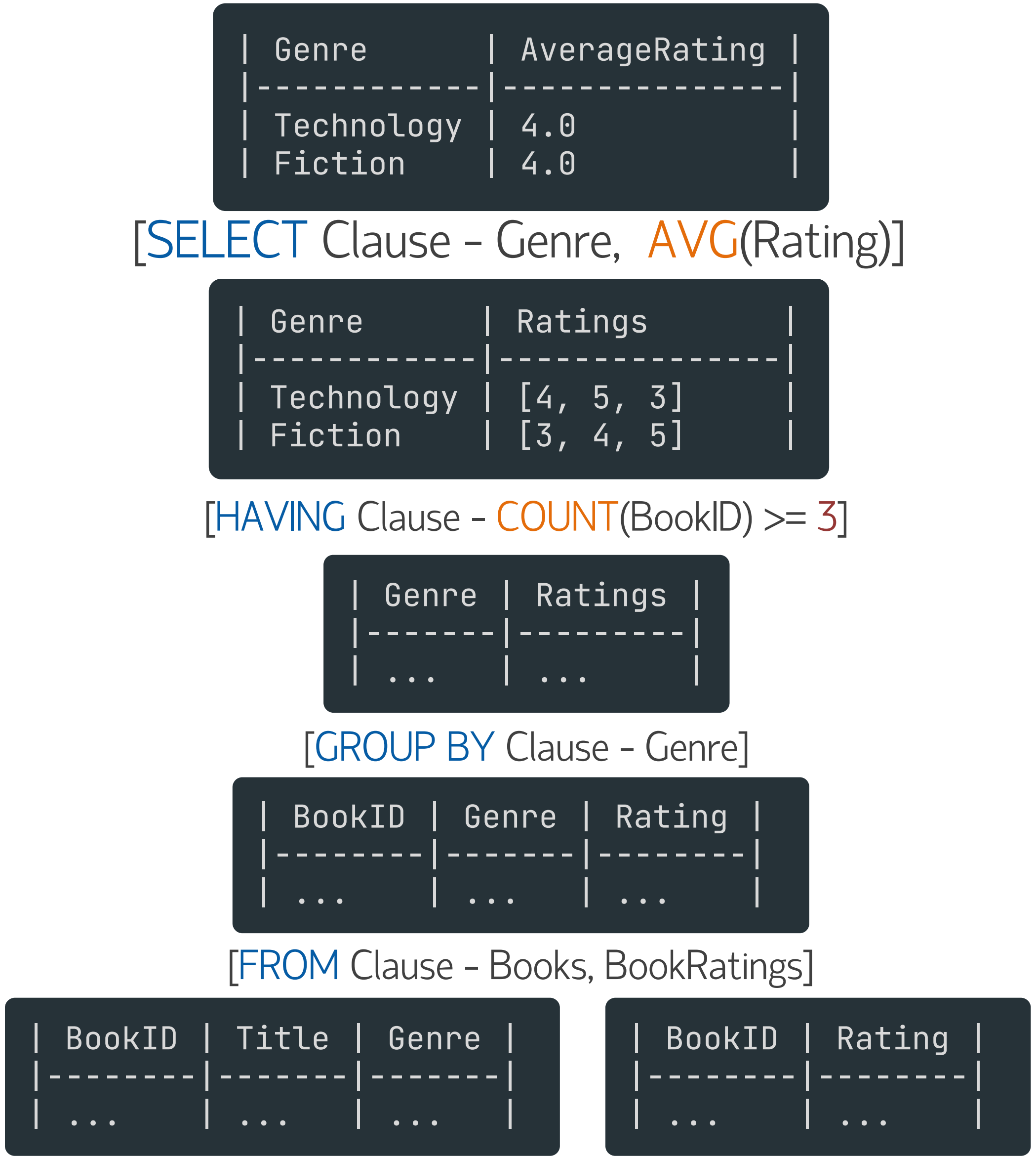
Underestanding Execution Order

FWGHOS Walk Through

- SQL Query:

```sql
SELECT Genre, AVG(Rating) AS AverageRating
FROM Books
INNER JOIN BookRatings
ON Books.BookID = BookRatings.BookID
GROUP BY Genre
HAVING COUNT(BookID) >= 3;
```

- **Propose:** To identify genres that are highly rated by readers (above a certain threshold).



```
| Genre      | AverageRating |
|------------|---------------|
| Technology | 4.0           |
| Fiction    | 4.0           |
```

[SELECT Clause - Genre, AVG(Rating)]

```
| Genre      | Ratings       |
|------------|---------------|
| Technology | [4, 5, 3]     |
| Fiction    | [3, 4, 5]     |
```

[HAVING Clause - COUNT(BookID) >= 3]

```
| Genre | Ratings |
|-------|---------|
| ...   | ...     |
```

[GROUP BY Clause - Genre]

```
| BookID | Genre | Rating |
|--------|-------|--------|
| ...    | ...   | ...    |
```

[FROM Clause - Books, BookRatings]

```
| BookID | Title | Genre |
|--------|-------|-------|
| ...    | ...   | ...   |
```

```
| BookID | Rating |
|--------|--------|
| ...    | ...    |
```

Self Joins

- Find all authors who published in both Eduction and Technology genres.
- **Source Table:** 'Books'

```
| BookID | Title                             | Author        | Genre      | PublishedYear |
|--------|-----------------------------------|---------------|------------|---------------|
| 101    | Journey Through SQL               | A. Coder      | Technology | 2015          |
| 102    | The History of Databases          | D. Base       | Education  | 2018          |
| 103    | Adventures in Coding              | P. Programmer | Fiction    | 2020          |
| 104    | Exploring Data Science            | A. Coder      | Science    | 2021          |
| 105    | Fundamentals of Database Design   | D. Base       | Technology | 2019          |
| ...    | ...                               | ...           | ...        | ...           |
```

- SQL Query:

```
SELECT b1.Author
FROM Books b1
INNER JOIN Books b2 ON b1.Author = b2.Author
WHERE b1.Genre = 'Education'
AND b2.Genre = 'Technology';
```

- Output Table:

```
| Author   |
| -------- |
| D. Base  |
```

- When a relation occurs twice in the FROM clause, we call it a self-join.
- A self-join is used to compare rows within the same table. It's particularly useful for analysing complex relationships within a single table.

Hand-On SQL Demonstration

- **Focus:** Utilising SQL for advanced data analysis in a library database.
- **Goal:** Demonstrate how to summarize and categorize data effectively using SQL's aggregation and grouping capabilities.

Example Query

- **Calculate Average Book Rating by Genre:**
  - SQL Query:

```sql
SELECT Genre, AVG(Rating) AS AverageRating
FROM Books b INNER JOIN BookRatings br
ON b.BookID = br.BookID
GROUP BY Genre;
```

  - **Purpose**: Show how SQL can be used to calculate aggregate data (average rating) and organize it by categories (genres).

Try It Yourself

1. Count Books in Each Genre:
   - **Task**: Write a query to count the number of books in each genre.
2. Find Maximum Rating in Each Genre:
   - **Task**: Modify the query to find the highest book rating in each genre.

Recap and Key Takeaways

- SQL's aggregation functions, along with GROUP BY, are pivotal for condensing extensive datasets into meaningful summaries. This capability is essential in extracting key insights such as trends, facilitating informed decisions in data analysis.
- The combination of structured querying with aggregation and grouping in SQL allows for tailored, precise data analysis. This approach underscores the relational model's efficiency in delivering organized and reliable insights, crucial for database management and business intelligence.

Example SQL Recap:

```
SELECT b.Genre, AVG(br.Rating) AS AverageRating
FROM Books b
INNER JOIN BookRatings br ON b.BookID = br.BookID
GROUP BY b.Genre;
```

- **Purpose:** Computes the average book ratings by genre.

Preparing for Subqueries:

- Next Lecture Preview:
  - Managing Complexities with **SQL Subqueries**.
  - Understanding **Subqueries** via **For-Each Sematics**.

Can you think of a scenario where advanced SQL functions (such as window functions) might be necessary to complement basic aggregation and grouping for more complex data analysis?