

Relational Databases

Aggregation and Grouping

8 February 2024

克明峻德，格物致知

Quick Recap: Joining Tables

Key Concepts:

- Joins enable linking data across tables, with various types: **Inner Join**: Matches rows between tables, **Left Join**: Includes all rows from the left table, and matched rows from the right, **Anti Left Join**: Shows rows from the left table without a match in the right. **Self Join**: Joins a table to itself for analyzing internal relationships.
- Joins often rely on Primary and Foreign Keys to ensure data consistency between related tables, maintaining relational integrity.

SQL Query Example


```
SELECT b.Title, br.ReturnDate
FROM Books b
INNER JOIN Borrow br
ON b.BookID = br.BookID;
```

- Nested-Loop Semantics

```
for each row1 in Books:
  for each row2 in Borrow:
    if (row1.BookID = row2.BookID):
      output (row1.Title, row2.ReturnDate)
```

- Purpose:** This query retrieves the titles of borrowed books along with their respective return dates.
- Output Table:**

| Title | ReturnDate |
|----------------------|------------|
| Journey Through SQL. | 2022-01-30 |
| Adventures in Coding | 2022-02-14 |



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

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

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

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.

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:

```
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:

```
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 |
| ... | ... |

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:

```
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]
```


Understanding 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).

Understanding 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(*) ≥ 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 |

Understanding 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(*) ≥ 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 |
|--------|--------|
| ... | ... |

Understanding 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(*) ≥ 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 |
|--------|--------|
| ... | ... |

Understanding 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(*) ≥ 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 |
|--------|--------|
| ... | ... |

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 Semantics.

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?

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:

```
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:** 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.