Relational Databases

Joining Tables

8 February 2024

克明峻德，格物致知

Quick Recap: Basic SQL Query

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

```sql
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 |
| ...                     | ...     |
```

Insurance | Credit cards | Loans

CAR | TRAVEL | HEALTH

I'm looking for single travel insurance for

Japan

08-05-2

START OVER

If you click on "Show my results" above, it means that you agree to our terms and privacy policies.

Google DISPLAY NETWORK

facebook Ads

Google Ads

#tiq by Etiqa Insurance

AXA

FWD insurance

MORE THAN 40 MILLION USERS HAVE FOUND WHAT THEY WERE LOOKING FOR.

ompare.

asiest

d what

Q: Which are our profitable travel insurance products?

A: To know this, we will need to create an end-to-end platform data, i.e. digital marketing, conversion (or click) and revenues.

direct asia insurance

ERGO

eTiQa Insurance

HL Assurance

Joins

Key to Linking Data Across Tables

- **Primary Role:** Joins in SQL are essential for merging data from different tables, enabling a comprehensive analysis of relational data.

Relationship with Foreign Keys

- **Foreign Keys:** Serve to define and enforce relationships between tables. While not mandatory for joins, they are commonly used to establish direct links, ensuring data integrity.
- **Without Foreign Keys:** Joins can still be performed on related columns, allowing flexibility in querying across tables even without strict foreign key constraints.

Enhancing Data Insights

- **Data Combination:** By joining tables, SQL queries can extract and combine relevant information from multiple sources, providing more in-depth insights.
- **Example:** Linking customer profiles with their transaction records to understand purchasing behaviors.

Inner Joins

Essential Tool in SQL Queries

- **Inner Join** is a basic SQL operation that merges rows from multiple tables based on a common column.

Inner Join Syntax

- SQL Query:

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

- This query joins the Books table (b) and the Borrow table (br) using the BookID as the common column.

- Books Table

```
| BookID | Title                    | Author         |
|--------|--------------------------|----------------|
| 101    | Journey Through SQL      | A. Coder       |
| 102    | The History of Databases | D. Base        |
| 103    | Adventures in Coding     | P. Programmer  |
| ...    | ...                      | ...            |
```

- Borrow Table

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

Inner Joins

Essential Tool in SQL Queries

- **Inner Join** is a basic SQL operation that merges rows from multiple tables based on a common column.

Inner Join Syntax

- SQL Query:

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

- This query joins the Books table (b) and the Borrow table (br) using the BookID as the common column.
- **Join Predicate:** b.BookID = br.BookID

- Books Table

```
| BookID | Title                     | Author         |
|--------|---------------------------|----------------|
| 101    | Journey Through SQL       | A. Coder       |
| 102    | The History of Databases  | D. Base        |
| 103    | Adventures in Coding      | P. Programmer  |
| ...    | ...                       | ...            |
```

- Borrow Table

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

Nested-Loop Semantics

Understanding Through Examples:

- Books Table:

```
| BookID | Title                     | Author          |
|--------|---------------------------|-----------------|
| 101    | Journey Through SQL       | A. Coder        |
| 102    | The History of Databases  | D. Base         |
| 103    | Adventures in Coding      | P. Programmer   |
| ...    | ...                       | ...             |
```

- Borrow Table:

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

- SQL Query:

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

Nested-Loop Semantics

Understanding Through Examples:

- Books Table:

```
| BookID | Title                     | Author         |
|--------|---------------------------|----------------|
| 101    | Journey Through SQL       | A. Coder       |
| 102    | The History of Databases  | D. Base        |
| 103    | Adventures in Coding      | P. Programmer  |
| ...    | ...                       | ...            |
```

- Borrow Table:

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

- SQL Query:

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

- Output Table:

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

Nested-Loop Semantics

Understanding Through Examples:

- Books Table:

```
| BookID | Title                     | Author          |
|--------|---------------------------|-----------------|
| 101    | Journey Through SQL       | A. Coder        |
| 102    | The History of Databases  | D. Base         |
| 103    | Adventures in Coding      | P. Programmer   |
| ...    | ...                       | ...             |
```

- Borrow Table:

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

- SQL Query:

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

- Output Table:

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

- *How does the SQL engine process this JOIN?*
- We compare every possible combination and filter the results that match.

Nested-Loop Semantics

Exploring How SQL Executes Joins:

- Books Table:

```
| BookID | Title                      | Author          |
|--------|----------------------------|-----------------|
| 101    | Journey Through SQL        | A. Coder        |
| 102    | The History of Databases   | D. Base         |
| 103    | Adventures in Coding       | P. Programmer   |
| ...    | ...                        | ...             |
```

- Borrow Table:

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

- SQL Query:

```sql
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)
```

- Output Table:

```
| Title                | ReturnDate |
|----------------------|------------|
```

Nested-Loop Semantics

Exploring How SQL Executes Joins:

- Books Table:

```
| BookID | Title                      | Author          |
|--------|----------------------------|-----------------|
| 101    | Journey Through SQL        | A. Coder        |
| 102    | The History of Databases   | D. Base         |
| 103    | Adventures in Coding       | P. Programmer   |
| ...    | ...                        | ...             |
```

- Borrow Table:

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

- SQL Query:

```sql
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)
```

- Output Table:

```
| Title                | ReturnDate |
|----------------------|------------|
| Journey Through SQL  | 2022-01-30 |
```

Nested-Loop Semantics

Exploring How SQL Executes Joins:

- Books Table:

```
| BookID | Title                     | Author          |
|--------|---------------------------|-----------------|
| 101    | Journey Through SQL       | A. Coder        |
| 102    | The History of Databases  | D. Base         |
| 103    | Adventures in Coding      | P. Programmer   |
| ...    | ...                       | ...             |
```

- Borrow Table:

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

- SQL Query:

```sql
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)
```

- Output Table:

```
| Title                | ReturnDate |
|----------------------|------------|
| Journey Through SQL  | 2022-01-30 |
```

Nested-Loop Semantics

Exploring How SQL Executes Joins:

- Books Table:

BookID	Title	Author
101	Journey Through SQL	A. Coder
102	The History of Databases	D. Base
103	Adventures in Coding	P. Programmer
...	...	...

- Borrow Table:

BookID	MemberID	BorrowDate	ReturnDate
101	501	2022-01-15	2022-01-30
103	502	2022-02-01	2022-02-14

- SQL Query:

```sql
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)
```

- Output Table:

Title	ReturnDate
Journey Through SQL	2022-01-30

Nested-Loop Semantics

Exploring How SQL Executes Joins:

- Books Table:

```
| BookID | Title                      | Author          |
|--------|----------------------------|-----------------|
| 101    | Journey Through SQL        | A. Coder        |
| 102    | The History of Databases   | D. Base         |
| 103    | Adventures in Coding       | P. Programmer   |
| ...    | ...                        | ...             |
```

- Borrow Table:

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

- SQL Query:

```sql
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)
```

- Output Table:

```
| Title                | ReturnDate |
|----------------------|------------|
| Journey Through SQL  | 2022-01-30 |
```

Nested-Loop Semantics

Exploring How SQL Executes Joins:

- Books Table:

```
| BookID | Title                      | Author          |
|--------|----------------------------|-----------------|
| 101    | Journey Through SQL        | A. Coder        |
| 102    | The History of Databases   | D. Base         |
| 103    | Adventures in Coding       | P. Programmer   |
| ...    | ...                        | ...             |
```

- Borrow Table:

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

- SQL Query:

```sql
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)
```

- Output Table:

```
| Title                | ReturnDate |
|----------------------|------------|
| Journey Through SQL  | 2022-01-30 |
```

Nested-Loop Semantics

Exploring How SQL Executes Joins:

- Books Table:

```
| BookID | Title                     | Author         |
|--------|---------------------------|----------------|
| 101    | Journey Through SQL       | A. Coder       |
| 102    | The History of Databases  | D. Base        |
| 103    | Adventures in Coding      | P. Programmer  |
| ...    | ...                       | ...            |
```

- Borrow Table:

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

- SQL Query:

```sql
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)
```

- Output Table:

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

Inner Joins: Implicit vs Explicit

Understanding Through Examples:

- Books Table:

```
| BookID | Title                      | Author         |
|--------|----------------------------|----------------|
| 101    | Journey Through SQL        | A. Coder       |
| 102    | The History of Databases   | D. Base        |
| 103    | Adventures in Coding       | P. Programmer  |
| ...    | ...                        | ...            |
```

- Borrow Table:

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

- Explicit Join Syntax:

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

- Implicit Join Syntax

```sql
SELECT b.Title, br.ReturnDate
FROM Books b, Borrow br
WHERE b.BookID = br.BookID;
```

- Output Table:
  - Both queries yield the same result.

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

Implicit vs. Explicit

- Both perform inner joins with different syntax: implicit uses a WHERE clause, while explicit uses JOIN ... ON.

LEFT JOIN

Inclusive Data Retrieval with Outer Joins

- **Books Table:**

```
| BookID | Title                    | Author         |
|--------|--------------------------|----------------|
| 101    | Journey Through SQL      | A. Coder       |
| 102    | The History of Databases | D. Base        |
| 103    | Adventures in Coding     | P. Programmer  |
| ...    | ...                      | ...            |
```

- **Borrow Table:**

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

- **SQL Query:**

```sql
SELECT b.Title
FROM Books b
LEFT JOIN Borrow br ON b.BookID = br.BookID;
```

- **Purpose:** Retrieves all books, regardless of whether they have been borrowed.
- **Output Table:**

```
| Title                    | ReturnDate |
|--------------------------|------------|
| Journey Through SQL      | 2022-01-30 |
| The History of Databases | NULL       |
| Adventures in Coding     | 2022-02-14 |
```

- **LEFT [OUTER] JOIN Concept:**
  - For books that have been borrowed, it shows their return date.
  - For books that have not been borrowed, the return date will be NULL.

ANTI LEFT JOIN

Identifying Records with No Match in Another Table

- **Books Table:**

```
| BookID | Title                    | Author         |
|--------|--------------------------|----------------|
| 101    | Journey Through SQL      | A. Coder       |
| 102    | The History of Databases | D. Base        |
| 103    | Adventures in Coding     | P. Programmer  |
| ...    | ...                      | ...            |
```

- **Borrow Table:**

```
| BookID | MemberID | BorrowDate | ReturnDate |
|--------|----------|------------|------------|
| 101    | 501      | 2022-01-15 | 2022-01-30 |
| 103    | 502      | 2022-02-01 | 2022-02-14 |
```

- **SQL Query:**

```sql
SELECT b.Title
FROM Books b
LEFT JOIN Borrow br ON b.BookID = br.BookID
WHERE br.BookID IS NULL;
```

- **Purpose:** Identifying Unborrowed Books.
- **Output Table:**

```
| Title                    |
|--------------------------|
| The History of Databases |
```

- **Critical for Data Analysis:**
  - Essential for identifying unlinked records, it offers insights into less popular or demanded items, completing the data picture.

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:**

```sql
SELECT Author
FROM Books
WHERE Genre = 'Technology'
AND Genre = 'Science';
```

- **Output Table:**

```
| Author    |
| --------- |
```

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:**

```sql
SELECT Author
FROM Books
WHERE Genre IN ('Education', 'Technology');
```

- **Output Table:**

Author
A. Coder
D. Base
D. Base

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:

```sql
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:** Understanding and applying SQL joins in a library database scenario.
- **Goal:** Showcase how SQL joins can combine data from different tables for comprehensive insights.

Example Query

- **Find Borrowed Book Titles and Return Dates:**
  - *SQL Query:*

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

  - Purpose: Demonstrates how INNER JOIN combines book titles with their borrowing details.

Try It Yourself

1. Find Books Borrowed During a Specific Year:
   - Task: Write a query to list titles of books that were borrowed in the year 2022.
2. List Books and Borrowers:
   - Task: Create a query to display the names of members alongside the titles of books they have borrowed.

Recap and Key Takeaways

- JOIN operations are crucial for merging and analyzing data across tables, enabling comprehensive insights and informed decision-making in relational databases.
- The JOIN condition can be equality based on a pair of columns from the 2 tables or other more complicated logic based on multiple columns, or even a self-join that compares columns within the same table.

Example SQL Recap:

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

- **Purpose:** This query retrieves the titles of borrowed books along with their respective return dates, linking book information with borrowing records.

Preparing for Aggregates and Grouping:

- Next Lecture Preview:
  - Explore further into SQL Aggregation Functions.
  - Examine **Relational Algebra** equivalencies.

Why is maintaining data integrity crucial in a database, and how do primary and foreign keys contribute to this process?