ECON485

JOINs in SQL

We will explain each JOIN type building upon the Library example.

1. JOIN Concept

In Structured Query Language (SQL), a JOIN clause is used to combine rows from two or more tables in a relational database based on a related column between them, typically a foreign key relationship. The JOIN operation produces a result set that includes columns from the participating tables, filtered by the join condition, θ.

Formally, given two relations (tables) R and S, with a join predicate θ (e.g., R.a = S.b), the JOIN can be defined as follows for common types:

- INNER JOIN: Returns only the rows where there is a match in both tables according to θ. Mathematically: $R \, X_\theta \, S = \{ t \mid t \in (R \, X \, S) \cap \theta(t) \}$, where R X S is the Cartesian product.
- LEFT (OUTER) JOIN: Returns all rows from the left table R, and matched rows from the right table S; unmatched rows in R are preserved with NULL values in columns from S.
- RIGHT (OUTER) JOIN: Symmetric to LEFT JOIN, preserving all rows from S and matching from R.
- FULL OUTER JOIN: Returns all rows from both tables, with NULLs where no match exists.
- CROSS JOIN: Produces the Cartesian product without a condition: R X S.

In set theory terms, an INNER JOIN resembles the intersection of sets based on the join condition, while OUTER JOINs are like unions that include non-matching elements with placeholders.

In relational algebra (the theoretical foundation of SQL), JOIN is directly inspired by the natural join operator X, which combines a Cartesian product with a selection predicate to filter matching tuples — SQL's JOIN clause is essentially a practical implementation of this algebraic operation.

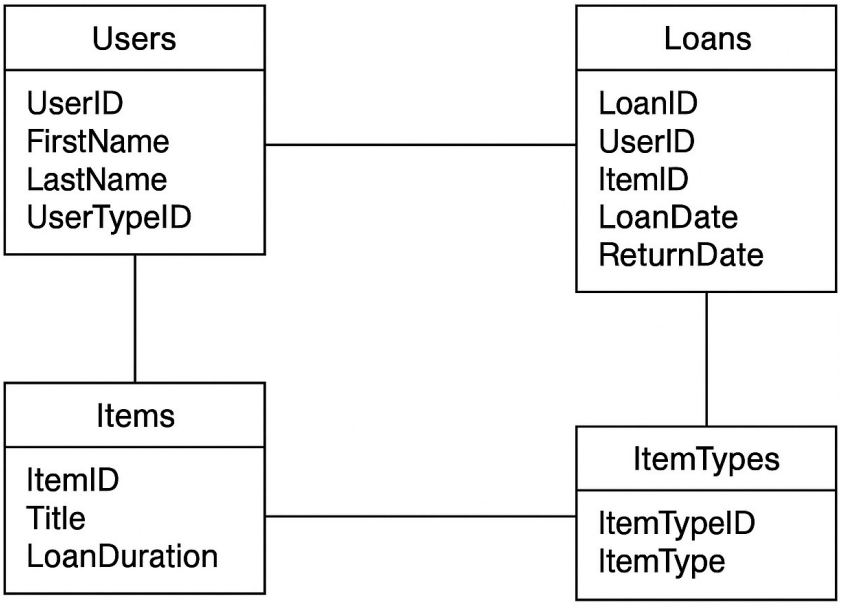

Figure 1: Simplified ER Diagram as a reminder for the table structure in the Library example (Items table does not have Authors in it, as well as Authors and Publishers tables are omitted).

2. INNER JOIN

Formal definition	Returns rows that have matching values in both tables according to the join condition.
Informal definition	“Only keep records that match on both sides.”
How keys are used	Matches Primary Key → Foreign Key pairs. Enforces relational integrity between normalized tables.
Level of Normalization Typically Expected	Appears at 2NF and above (when entities are separated into related tables).
Common Uses in Practice	Used to combine core entities, e.g., facts with their dimensions.
Typical Misuses / Pitfalls	Forgetting WHERE conditions, causing fewer rows than expected; joining on non-key columns can cause duplicates.
Example SQL (Library Schema)	SELECT u.FirstName, i.Title, l.LoanDate FROM Loans l JOIN Users u ON u.UserID = l.UserID JOIN Items i ON i.ItemID = l.ItemID;

3. LEFT (OUTER) JOIN

Formal definition	Returns all rows from the left table and the matched rows from the right table. Non-matching right-side rows return NULLs.
Informal definition	“Show everything from the left, even if there’s no match.”
How keys are used	Uses **PK–FK** links but allows NULLs when no match exists on the FK side.
Level of Normalization Typically Expected	Often used in **3NF and beyond** to report missing or optional relationships.
Common Uses in Practice	To find users with no loans, items never borrowed, or optional metadata.
Typical Misuses / Pitfalls	Treating NULLs as zero; assuming NULLs mean missing data rather than “no relationship.”
Example SQL (Library Schema)	SELECT u.UserID, u.FirstName, COUNT(l.LoanID) AS LoanCount FROM Users u LEFT JOIN Loans l ON u.UserID = l.UserID GROUP BY u.UserID, u.FirstName;

4. RIGHT (OUTER) JOIN

Formal definition	Returns all rows from the right table and matched rows from the left.
Informal definition	“Like LEFT JOIN but from the other side.”
How keys are used	Same as LEFT JOIN but reversed order of tables.
Level of Normalization Typically Expected	Same as LEFT JOIN; less common.
Common Uses in Practice	Rarely used; often replaced by LEFT JOIN with reversed order.
Typical Misuses / Pitfalls	Reduces clarity; may confuse readers.
Example SQL (Library Schema)	SELECT i.Title, l.LoanID FROM Loans l RIGHT JOIN Items i ON i.ItemID = l.ItemID;

5. FULL (OUTER) JOIN

Formal definition	Returns all rows when there is a match in one of the tables; unmatched rows from both sides are included.
Informal definition	“Combine everything, even if only one side has data.”
How keys are used	Combines PK–FK relations and shows NULLs for both unmatched sides.
Level of Normalization Typically Expected	More common in **data warehouse** or **reporting** contexts than in 3NF OLTP.
Common Uses in Practice	Merging datasets, audit comparisons, synchronization.
Typical Misuses / Pitfalls	Can create NULL-heavy datasets and be computationally expensive.
Example SQL (Library Schema)	SELECT u.FirstName, l.LoanDate FROM Users u FULL JOIN Loans l ON u.UserID = l.UserID;

6. CROSS JOIN

Formal definition	Returns the Cartesian product of both tables (all possible combinations).
Informal definition	“Every row meets every other row.”
How keys are used	No key relationship; every row in table A is paired with every row in table B.
Level of Normalization Typically Expected	Appears in **1NF contexts** when data isn’t yet relationally meaningful.
Common Uses in Practice	Generating test data, combinations (e.g., seat × day).
Typical Misuses / Pitfalls	Forgetting WHERE clause after CROSS JOIN leads to exponential row explosion.

Example SQL (Library Schema)	`SELECT i.Title, d.DayName`
	`FROM Items I`
	`CROSS JOIN DaysOfWeek d;`
	
	`SELECT i.Title, d.DayName`
	`FROM Items i`
	`CROSS JOIN DaysOfWeek d`
	`WHERE d.DayName IN ('Monday', 'Tuesday') AND i.Available = TRUE;`
	
	`SELECT i.Title, d.DayName`
	`FROM Items i`
	`CROSS JOIN DaysOfWeek d`
	`WHERE (i.ItemType = 'Multimedia' AND d.DayName IN ('Saturday','Sunday')) OR (i.ItemType <> 'Multimedia' AND d.DayName NOT IN ('Saturday','Sunday'));`

7. SELF JOIN

Formal definition	Joins a table to itself using aliases to represent two roles of the same entity.
Informal definition	“Compare rows in the same table.”
How keys are used	Uses the **same PK–FK relationship** within one table (e.g., hierarchical parent-child).
Level of Normalization Typically Expected	Used when the schema is at least **3NF**, separating relationships but still self-referential.
Common Uses in Practice	Finding duplicates, hierarchies (e.g., same author, same title).
Typical Misuses / Pitfalls	Ambiguous aliases; forgetting to limit results.
Example SQL (Library Schema)	SELECT a.Title, b.Title FROM Items a JOIN Items b ON a.AuthorID = b.AuthorID AND a.ItemID <> b.ItemID;

8. SEMI JOIN (with EXISTS)

Formal definition	Returns rows from the left table where a match exists in the right table, but doesn’t return right table data.
Informal definition	“Show me entities that have something on the other side.”
How keys are used	Uses PK–FK logic inside an EXISTS subquery.
Level of Normalization Typically Expected	Appears in 3NF/BCNF normalized models where you filter entities by relationships.
Common Uses in Practice	Finding rows who have any relationships with other rows (ie. users with active loans)
Typical Misuses / Pitfalls	Misunderstanding it as INNER JOIN; duplicates are not produced.
Example SQL (Library Schema)	SELECT u.* FROM Users u WHERE EXISTS (SELECT 1 FROM Loans l WHERE l.UserID = u.UserID);

9. ANTI JOIN (via NOT EXISTS / LEFT JOIN .. IS NULL)

Formal definition	Returns rows from the left table with no matching rows in the right.
Informal definition	“Show me what’s missing on the other side.”
How keys are used	Checks absence of FK reference.
Level of Normalization Typically Expected	Common in **3NF reports** to verify data completeness.
Common Uses in Practice	Identifying rows without any relationships ever with other rows (ie. items never loaned, users never borrowed.)
Typical Misuses / Pitfalls	NULL comparison pitfalls; must handle NULL-safe logic.
Example SQL (Library Schema)	SELECT i.* FROM Items I LEFT JOIN Loans l ON i.ItemID = l.ItemID WHERE l.LoanID IS NULL;

10. JOINS and NFs

Based on your normal form:

- 1NF: Data is tabular; joins are rare or artificial (CROSS JOIN for generation).
- 2NF: Functional dependencies separated; joins become necessary between core tables.
- 3NF/BCNF: Dimension/fact joins dominate (INNER and LEFT JOINs).
- Higher Normal Forms: Rely heavily on JOINs for reconstruction of logical entities.