**1. INNER JOIN**

**Returns rows that have matching values in both tables.**

**SELECT s.StudentID, s.FullName, f.FieldName**

**FROM Students s**

**INNER JOIN Fields f ON s.FieldID = f.FieldID;**

**Shows only students who have a valid FieldID in the Fields table.**

**2. LEFT JOIN (LEFT OUTER JOIN)**

**Returns all rows from the left table (Students), even if there is no match in Fields.**

**SELECT s.StudentID, s.FullName, f.FieldName**

**FROM Students s**

**LEFT JOIN Fields f ON s.FieldID = f.FieldID;**

**If a student has no field, FieldName will be NULL.**

**3. RIGHT JOIN (RIGHT OUTER JOIN)**

**Returns all rows from the right table (Fields), even if there are no students in that field.**

**SELECT s.StudentID, s.FullName, f.FieldName**

**FROM Students s**

**RIGHT JOIN Fields f ON s.FieldID = f.FieldID;**

**Shows all fields, even if no students are enrolled in them.**

**4. FULL JOIN (FULL OUTER JOIN)**

**Returns all rows from both tables, with NULLs where there is no match.**

**SELECT s.StudentID, s.FullName, f.FieldName**

**FROM Students s**

**FULL JOIN Fields f ON s.FieldID = f.FieldID;**

**Shows all students and all fields (with NULL values if no match).**

**5. CROSS JOIN**

**Returns all possible combinations (Cartesian product).  
⚠️ Be careful: if you have 10 students and 10 fields → result = 100 rows.**

**SELECT s.FullName, f.FieldName**

**FROM Students s**

**CROSS JOIN Fields f;**

**6. SELF JOIN**

**A table joined with itself. (Not directly useful for Students + Fields, but an example:)**

**SELECT s1.FullName AS Student1, s2.FullName AS Student2**

**FROM Students s1**

**INNER JOIN Students s2 ON s1.FieldID = s2.FieldID**

**WHERE s1.StudentID <> s2.StudentID;**

**Finds students who are in the same field.**

**7. JOIN with Filtering**

**Example: find Computer Science students only:**

**SELECT s.FullName, f.FieldName**

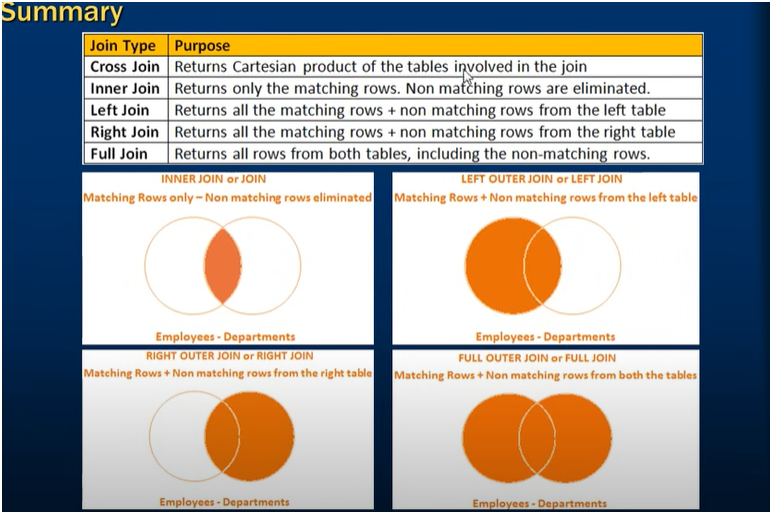
**FROM Students s**

**INNER JOIN Fields f ON s.FieldID = f.FieldID**

**WHERE f.FieldName = 'Computer Science';**

**So in summary:**

* **INNER JOIN → Matching rows only**
* **LEFT JOIN → All from left + matches**
* **RIGHT JOIN → All from right + matches**
* **FULL JOIN → All from both sides**
* **CROSS JOIN → All combinations**
* **SELF JOIN → A table joined to itself**

****

**Advanced or Intelligent joins in SQL**

**1. JOIN with Aggregates (GROUP BY + JOIN)**

**You can join tables and also calculate statistics.**

**-- Count students in each field**

**SELECT f.FieldName, COUNT(s.StudentID) AS StudentCount**

**FROM Fields f**

**LEFT JOIN Students s ON f.FieldID = s.FieldID**

**GROUP BY f.FieldName;**

**Even fields with zero students will show up because of LEFT JOIN.**

**2. JOIN with Subquery / Derived Table**

**Join with a subquery that precomputes something.**

**-- Find students who are in the largest field**

**SELECT s.FullName, f.FieldName**

**FROM Students s**

**INNER JOIN (**

**SELECT TOP 1 FieldID**

**FROM Students**

**GROUP BY FieldID**

**ORDER BY COUNT(\*) DESC**

**) bigField ON s.FieldID = bigField.FieldID**

**INNER JOIN Fields f ON s.FieldID = f.FieldID;**

**Shows students in the most populated field.**

**3. SELF JOIN (Hierarchical Data)**

**Useful for organizational or mentor/student style relations.**

**-- Suppose Students table has MentorID column (FK to StudentID)**

**SELECT s.FullName AS Student, m.FullName AS Mentor**

**FROM Students s**

**LEFT JOIN Students m ON s.MentorID = m.StudentID;**

**Joins a table to itself to show mentorship relationships.**

**4. APPLY (CROSS APPLY & OUTER APPLY)**

**APPLY is like a smart join with a subquery or table-valued function.**

**-- Find the most recent student in each field**

**SELECT f.FieldName, s.StudentID, s.FullName, s.BirthDate**

**FROM Fields f**

**CROSS APPLY (**

**SELECT TOP 1 \***

**FROM Students s**

**WHERE s.FieldID = f.FieldID**

**ORDER BY s.BirthDate DESC**

**) s;**

**CROSS APPLY executes subquery per row of Fields.  
OUTER APPLY would also return fields with no students.**

**5. JOIN with CASE (Conditional Joins)**

**Sometimes you join differently based on conditions.**

**-- Show field name, if missing write 'Not Assigned'**

**SELECT s.FullName,**

**CASE WHEN f.FieldName IS NULL THEN 'Not Assigned' ELSE f.FieldName END AS Field**

**FROM Students s**

**LEFT JOIN Fields f ON s.FieldID = f.FieldID;**

**6. FULL JOIN + WHERE (Symmetric Difference)**

**Find mismatched rows between two tables.**

**-- Students without a field OR fields without students**

**SELECT s.FullName, f.FieldName**

**FROM Students s**

**FULL JOIN Fields f ON s.FieldID = f.FieldID**

**WHERE s.StudentID IS NULL OR f.FieldID IS NULL;**

**This highlights inconsistencies.**

**7. Window Functions with Joins**

**Powerful way to rank or calculate over partitions.**

**-- Find oldest student in each field**

**SELECT f.FieldName, s.FullName, s.BirthDate**

**FROM Students s**

**INNER JOIN Fields f ON s.FieldID = f.FieldID**

**WHERE s.BirthDate = (**

**SELECT MIN(BirthDate)**

**FROM Students s2**

**WHERE s2.FieldID = s.FieldID**

**);**

**Or with ROW\_NUMBER:**

**SELECT \***

**FROM (**

**SELECT s.StudentID, s.FullName, f.FieldName,**

**ROW\_NUMBER() OVER (PARTITION BY s.FieldID ORDER BY s.BirthDate ASC) AS rn**

**FROM Students s**

**INNER JOIN Fields f ON s.FieldID = f.FieldID**

**) t**

**WHERE rn = 1;**

**8. Intelligent Joins with CTEs**

**Common Table Expressions make complex joins more readable.**

**;WITH FieldCounts AS (**

**SELECT FieldID, COUNT(\*) AS StudentCount**

**FROM Students**

**GROUP BY FieldID**

**)**

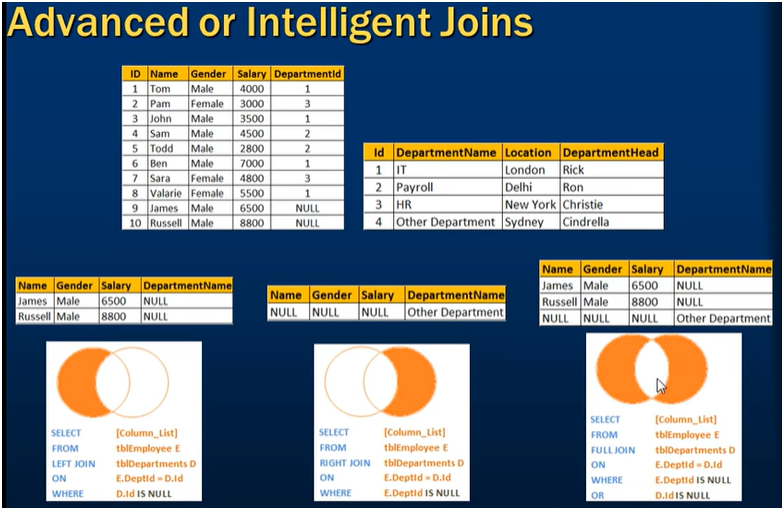
**SELECT f.FieldName, fc.StudentCount**

**FROM Fields f**

**LEFT JOIN FieldCounts fc ON f.FieldID = fc.FieldID;**

**Summary of advanced join techniques:**

* **Aggregates + Joins → Statistics per group.**
* **Subqueries + Joins → More selective joining.**
* **SELF JOIN → Hierarchies.**
* **APPLY (CROSS/OUTER APPLY) → Smart row-by-row joins.**
* **Conditional Joins with CASE.**
* **FULL JOIN + WHERE → Detect mismatches.**
* **Window Functions → Ranking, first/last, analytics.**
* **CTEs → Cleaner complex joins.**

****

**SELF JOIN IN SQL SERVER**

A **self join** is when a table is joined with itself.  
It’s useful when you have **hierarchical data** (like manager → employee, mentor → student, parent → child).

Technically, it’s just a **normal join** where the same table is given two different **aliases**.

## Example 1: Students and Mentors

Let’s add a MentorID column to your **Students** table:

ALTER TABLE Students ADD MentorID INT NULL;

Now, MentorID refers to another student (StudentID) from the same table.

### Query with SELF JOIN:

SELECT s.StudentID, s.FullName AS Student, m.FullName AS Mentor

FROM Students s

LEFT JOIN Students m ON s.MentorID = m.StudentID;

Output:

| **StudentID** | **Student** | **Mentor** |
| --- | --- | --- |
| 1 | Alice Johnson | NULL |
| 2 | Bob Smith | Alice Johnson |
| 3 | Charlie Davis | Bob Smith |

## Example 2: Students in Same Field

Let’s say we want to find pairs of students who are in the same field.

SELECT s1.FullName AS Student1, s2.FullName AS Student2, f.FieldName

FROM Students s1

INNER JOIN Students s2 ON s1.FieldID = s2.FieldID AND s1.StudentID < s2.StudentID

INNER JOIN Fields f ON s1.FieldID = f.FieldID;

Output (if Alice & Bob are both in Computer Science):

| **Student1** | **Student2** | **FieldName** |
| --- | --- | --- |
| Alice Johnson | Bob Smith | Computer Science |
| Charlie Davis | Diana Evans | Information Tech |

## Example 3: Organizational Chart

If you had ManagerID in a Employees table:

SELECT e.FullName AS Employee, m.FullName AS Manager

FROM Employees e

LEFT JOIN Employees m ON e.ManagerID = m.EmployeeID;

This is the classic **hierarchical self join**.

**Summary**:

* A self join is just a normal join, but the table is used twice with different aliases.
* Common use cases:
  + Finding parent-child relationships (manager → employee, mentor → student).
  + Comparing rows within the same table (students in the same field).
  + Building organizational charts or hierarchies.