Introduction to SQL Joins

SQL **joins** allow you to retrieve data from multiple related tables based on a common column. In relational databases, data is often stored in **separate tables** to maintain normalization, so joins are essential for retrieving meaningful information.

There are several types of SQL joins:

1. **INNER JOIN** – Returns only matching records.
2. **LEFT JOIN (LEFT OUTER JOIN)** – Returns all records from the left table and matching ones from the right.
3. **CROSS JOIN** – Produces a Cartesian product of two tables.
4. **SELF JOIN** – Joins a table to itself.

Note: SQLite **does not support RIGHT JOIN or FULL OUTER JOIN** directly.

Sample Database Setup

Before diving into joins, let's create and populate **sample tables**.

Creating Sample Tables

```
PRAGMA foreign_keys = ON;

DROP TABLE IF EXISTS Employees;
DROP TABLE IF EXISTS Departments;

CREATE TABLE Departments (
    dept_id INTEGER PRIMARY KEY,
    dept_name TEXT UNIQUE NOT NULL
);

CREATE TABLE Employees (
    emp_id INTEGER PRIMARY KEY,
    emp_name TEXT NOT NULL,
    department_id INTEGER,
    FOREIGN KEY (department_id) REFERENCES Departments(dept_id) ON DELETE SET
NULL
);
```

Adding Foreign Keys in SQLite3

In SQLite, **foreign keys** ensure that the referenced value exists in the parent table before being inserted into the child table. However, **foreign key constraints are disabled by default in SQLite**, so they need to be enabled explicitly.

Before executing any queries, ensure that foreign keys are enabled:

Explanation of Foreign Key Constraints

- **department_id INTEGER** in the Employees table is a foreign key.
- **FOREIGN KEY (department_id) REFERENCES Departments(dept_id)** ensures that any department_id in Employees **must exist in Departments**.
- **ON DELETE SET NULL** ensures that if a department is deleted, the department_id in Employees is set to NULL instead of being deleted.

Inserting Data (with Foreign Key Constraints)

Now, let's insert **valid data** into both tables.

```
INSERT INTO Departments (dept_id, dept_name) VALUES
(1, 'HR'),
(2, 'IT'),
(3, 'Finance'),
(4, 'Marketing');

INSERT INTO Employees (emp_id, emp_name, department_id) VALUES
(1, 'Alice', 1),      -- HR
(2, 'Bob', 2),        -- IT
(3, 'Charlie', 1),    -- HR
(4, 'David', NULL),   -- No department
(5, 'Eve', 3);        -- Finance
```

Enforcing Referential Integrity

Now, let's **test** what happens if we try to insert an invalid department ID.

```
INSERT INTO Employees (emp_id, emp_name, department_id) VALUES
(6, 'Frank', 99);
```

`This will fail because dept_id 99 does not exist`

Error: `FOREIGN KEY constraint failed`
SQLite **prevents** inserting `department_id = 99` because **dept_id 99 does not exist in Departments**.

Testing ON DELETE Behaviour

Scenario: Deleting a Department

```
DELETE FROM Departments WHERE dept_id = 1;
```

Since we used **ON DELETE SET NULL**, employees **Alice and Charlie** will now have department_id = NULL, rather than being deleted.

Verifying the Change

```
SELECT * FROM Employees;
```

Updated Employees Table

emp_id	emp_name	department_id
1	Alice	NULL
2	Bob	2
3	Charlie	NULL
4	David	NULL
5	Eve	3

SQL Join Types

Now, let's explore each type of join.

1. INNER JOIN

An **INNER JOIN** returns only records where there is a **match** in both tables.

Example Query

```
SELECT e.emp_id, e.emp_name, d.dept_name
FROM Employees e
INNER JOIN Departments d
ON e.department_id = d.dept_id;
```

Output

emp_id	emp_name	dept_name
1	Alice	HR
2	Bob	IT
3	Charlie	HR

Explanation

- Employees **Alice, Bob, and Charlie** appear because they have matching department IDs.
- **David (4)** is missing because his department_id is NULL.
- **Marketing (dept_id 4)** is missing because no employee belongs to it.

2. LEFT JOIN (LEFT OUTER JOIN)

A **LEFT JOIN** returns all records from the **left table** (Employees) and matching records from the right table (Departments). If no match is found, NULL values are returned.

Example Query

```
SELECT e.emp_id, e.emp_name, d.dept_name
FROM Employees e
LEFT JOIN Departments d
ON e.department_id = d.dept_id;
```

Output

emp_id	emp_name	dept_name
1	Alice	HR
2	Bob	IT
3	Charlie	HR
4	David	NULL
5	Eve	Finance

Explanation

- All **employees** are included.
- **David** has no department, so dept_name is NULL.

3. RIGHT JOIN (RIGHT OUTER JOIN)

A **RIGHT JOIN** returns all records from the **right table** (Departments) and matching records from the left table (Employees).

Example Query

```
-- Right Join
SELECT e.emp_id, e.emp_name, d.dept_name
FROM Employees e
RIGHT JOIN Departments d
ON e.department_id = d.dept_id;
```

Output

emp_id	emp_name	dept_name
1	Alice	HR
2	Bob	IT
3	Charlie	HR
5	Eve	Finance
NULL	NULL	Marketing

Explanation

The **Marketing department** is included, even though no employee is assigned to it.

Note: RIGHT JOIN is not supported in SQLite. Use **LEFT JOIN by swapping table positions**.

4 CROSS JOIN

A **CROSS JOIN** produces the **Cartesian Product**, meaning every row from the first table joins with every row from the second table.

Example Query

```
-- Cross Join
SELECT e.emp_name, d.dept_name
FROM Employees e
CROSS JOIN Departments d;
```

Output (if 5 employees and 4 departments)

Total **5 × 4 = 20** rows.