# Lenguaje SQL : Joins

Los `JOIN` son operaciones fundamentales en bases de datos relacionales, utilizadas para combinar filas de dos o más tablas en función de una columna relacionada entre ellas. Veamos ejemplos básicos utilizando dos tablas imaginarias: `students` y `grades`.

Imagina las siguientes tablas:

**students**:

| ID | nombre |
|---|---|
| 1 | Roberto |
| 2 | Alicia |
| 3 | Ana |
| 4 | Francisco |

**grades**:

| studentID | materia | calificacion |
|---|---|---|
| 1 | Matematicas | 8 |
| 2 | Matematicas | 10 |
| 1 | Literatura | 9 |
| 3 | Literatura | 9 |
| 8 | Idiomas | 9 |

---

▼ **Actividad 2:**

NOTA: Crea la tablas anteriores con los siguientes caracteristicas.

Nombre de tabla 1: **students**

| Nombre de columna | Tipo de dato |
|---|---|
| ID | INTEGER |
| nombre | TEXT |

Nombre de tabla 2: grades

| Nombre de columna | Tipo de dato |
|---|---|
| studentID | INTEGER |
| materia | TEXT |
| calificacion | INTEGER |

Inserta los valores de las tablas de students y grade.

---

1. **INNER JOIN**:

   Este tipo de join retorna filas cuando hay al menos una coincidencia en ambas tablas.

```
SELECT students.Nombre, grades.Materia, grades.Calificacion
FROM students
INNER JOIN grades ON students.ID = grades.studentID;
```

   Resultado:

| Nombre | Materia | Calificacion |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |

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

   Retorna todas las filas de la tabla izquierda y las coincidentes de la tabla derecha. Si no hay coincidencia, el resultado es NULL.

```
SELECT students.Nombre, grades.Materia, grades.Calificacion
FROM students
LEFT JOIN grades ON students.ID = grades.studentID;
```

Resultado:

| Name | Subject | Grade |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |

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

   Retorna todas las filas de la tabla derecha y las coincidentes de la tabla izquierda. Si no hay coincidencia, el resultado es NULL.

```
SELECT students.Nombre, grades.Materia, grades.Calificacion
FROM students
RIGHT JOIN grades ON students.ID = grades.studentID;
```

   Resultado (similar al anterior porque estamos usando las mismas tablas y datos):

| Name | Subject | Grade |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |

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

   Retorna filas cuando hay una coincidencia en una de las tablas. Es decir, combina los resultados de LEFT y RIGHT JOIN.

```
SELECT students.Name, grades.Subject, grades.Grade
FROM students
FULL JOIN grades ON students.ID = grades.studentID;
```

Nota: SQLite no soporta directamente `FULL OUTER JOIN`, pero puedes emularlo con una combinación de `LEFT JOIN` y `UNION`.

Estos son ejemplos básicos de los tipos de JOINs. Hay variaciones y complejidades adicionales que puedes explorar según las necesidades de tu consulta.

| Name | Subject | Grade |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |

5. Dado que SQLite no soporta `FULL OUTER JOIN` de manera nativa, pero se puede emular con una combinación de `LEFT JOIN` y `UNION`.

   Para el mismo conjunto de datos de las tablas `students` y `grades`, aquí te dejo cómo podrías hacer un `FULL OUTER JOIN` en SQLite:

```
-- Emular FULL OUTER JOIN en SQLite

-- Primero hacemos un LEFT JOIN
SELECT students.Name, grades.Subject, grades.Grade
FROM students
LEFT JOIN grades ON students.ID = grades.studentID

UNION

-- Luego un LEFT JOIN pero con el orden de las tablas invertido,
-- para cubrir las filas que no fueron incluidas en el primer JOIN.
SELECT students.Name, grades.Subject, grades.Grade
FROM grades
LEFT JOIN students ON grades.studentID = students.ID
WHERE students.ID IS NULL;
```

| Name | Subject | Grade |
|---|---|---|
| | | |
| | | |
| | | |

| Name | Subject | Grade |
| --- | --- | --- |
| | | |
| | | |

El primer `LEFT JOIN` devuelve todas las filas de `students` y las coincidencias de `grades`. El segundo `LEFT JOIN` (con el orden de las tablas invertido) devuelve las filas de `grades` que no tienen coincidencias en `students`. La cláusula `WHERE students.ID IS NULL` asegura que solo obtengamos las filas de `grades` que no tienen correspondencia.

El resultado combinado de estos dos `LEFT JOIN` nos da un efecto similar al `FULL OUTER JOIN`.

# CROSS JOIN

El siguientes ejercicios son utilizando las tablas de Students y Grades.

```
SELECT s.ID, s.nombre, g.materia, g.calificacion
FROM students s
CROSS JOIN grades g;
```

Este SQL producirá un producto cartesiano de ambas tablas. Por lo que cada estudiante se listará con cada materia y calificación, independientemente de si tomó esa materia o no.

> ¿Cuantos registros resultan?

> ¿Cuantos registros resultan del cruce de una tabla de 10millones de registros con otra de 5 millones de registros?

💡 Es importante tener cuidado al usar **Cross Join** ya que puede producir un gran número de resultados, especialmente si las tablas involucradas son grandes.

# UNION

Crear y completar las tablas proporcionadas

1. **Crear y completar la tabla adicional de otra escuela**:

```
CREATE TABLE external_students (
    ID INTEGER PRIMARY KEY,
    nombre TEXT
);

INSERT INTO external_students (ID, nombre) VALUES
(1, 'Roberto'),
(2, 'Elena'),
(3, 'Luis'),
(4, 'Carmen');

CREATE TABLE external_grades (
    studentID INTEGER,
    materia TEXT,
    calificacion INTEGER,
    FOREIGN KEY(studentID) REFERENCES other_school_students(ID)
);

INSERT INTO external_grades (studentID, materia, calificacion) VALUES
(1, 'Matemáticas', 82),
(2, 'Matemáticas', 91),
(3, 'Historia', 78),
(4, 'Historia', 88);
```

1. **Ejemplo de UNION**:

Si queremos obtener una lista de todos los estudiantes de ambas escuelas sin repetir, podemos utilizar `UNION`:

```
SELECT nombre FROM students
UNION
SELECT nombre FROM external_students;
```

| Nombre |
| --- |
| |
| |
| |
| |
| |
| |
| |
| |
| |

Con `UNION`, los nombres duplicados serán eliminados, así que si hubiera un 'Juan' en ambas escuelas, sólo aparecería una vez en el resultado.

Si quisiéramos incluir los nombres duplicados, podríamos usar `UNION ALL`:

```
SELECT nombre FROM students
UNION ALL
SELECT nombre FROM external_students;
```

| Nombre |
| --- |
| |
| |
| |
| |
| |
| |
| |
| |
| |

Estos ejemplos demuestran cómo se pueden combinar resultados de diferentes tablas con `UNION`. Puede ser especialmente útil cuando se trabaja con bases de datos que tienen estructuras similares pero que representan diferentes entidades o contextos.