JOINs en PostgreSQL

Guía Completa Explicada con Ejemplos

Documento Técnico

14 de enero de 2026

Índice

1. Introducción **3**

2. Tablas de Ejemplo **3**

3. INNER JOIN **3**
3.1. Concepto . . . 3
3.2. Ejemplo . . . 3

4. LEFT JOIN **3**
4.1. Concepto . . . 3
4.2. Ejemplo . . . 4

5. RIGHT JOIN **4**
5.1. Concepto . . . 4
5.2. Ejemplo . . . 4

6. FULL OUTER JOIN **4**
6.1. Concepto . . . 4
6.2. Ejemplo . . . 4

7. CROSS JOIN **4**
7.1. Concepto . . . 4
7.2. Ejemplo . . . 5

8. JOIN usando USING **5**
8.1. Concepto . . . 5
8.2. Ejemplo . . . 5

9. JOIN sin Claves Primarias ni Foráneas **5**
9.1. Ejemplo válido . . . 5

10. Problema de Duplicados **5**

11. JOIN vs WHERE (Error común) **6**
11.1. Incorrecto . . . 6
11.2. Correcto . . . 6

12. JOIN y el Optimizador **6**
12.1. EXPLAIN ANALYZE . . . 6

13.Buenas Prácticas **6**

14.Resumen Final **6**

1. Introducción

Un **JOIN** permite combinar filas de dos o más tablas basándose en una condición lógica. En PostgreSQL, los JOINs son extremadamente flexibles y **no requieren obligatoriamente claves primarias ni foráneas**.

Las claves afectan a la **integridad y optimización**, pero no a la validez sintáctica del JOIN.

2. Tablas de Ejemplo

```
CREATE TABLE users (
    id INT PRIMARY KEY,
    username TEXT
);

CREATE TABLE orders (
    id INT PRIMARY KEY,
    user_id INT,
    amount NUMERIC
);
```

Estas tablas se usarán en todos los ejemplos.

3. INNER JOIN

3.1. Concepto

Devuelve solo filas donde la condición de unión se cumple en ambas tablas.

3.2. Ejemplo

```
SELECT u.username, o.amount
FROM users u
INNER JOIN orders o
ON u.id = o.user_id;
```

Explicación:

- Solo aparecen usuarios que tengan pedidos.
- Si no hay coincidencia, la fila se descarta.

Equivale a escribir simplemente `JOIN`.

4. LEFT JOIN

4.1. Concepto

Devuelve todas las filas de la tabla izquierda y las coincidencias de la derecha. Si no hay coincidencia, devuelve `NULL`.

4.2. Ejemplo

```
SELECT u.username, o.amount
FROM users u
LEFT JOIN orders o
ON u.id = o.user_id;
```

Explicación:

- Todos los usuarios aparecen.
- Usuarios sin pedidos muestran `NULL`.

Muy usado en reportes y dashboards.

5. RIGHT JOIN

5.1. Concepto

Es simétrico al LEFT JOIN, pero prioriza la tabla derecha.

5.2. Ejemplo

```
SELECT u.username, o.amount
FROM users u
RIGHT JOIN orders o
ON u.id = o.user_id;
```

Nota: En la práctica se evita, ya que siempre puede reescribirse como LEFT JOIN.

6. FULL OUTER JOIN

6.1. Concepto

Devuelve todas las filas de ambas tablas, coincidan o no.

6.2. Ejemplo

```
SELECT u.username, o.amount
FROM users u
FULL JOIN orders o
ON u.id = o.user_id;
```

Resultado:

- Usuarios sin pedidos → columnas de orders en NULL
- Pedidos sin usuario → columnas de users en NULL

Útil para detectar inconsistencias.

7. CROSS JOIN

7.1. Concepto

Producto cartesiano: combina todas las filas con todas.

7.2. Ejemplo

```
SELECT u.username, o.amount
FROM users u
CROSS JOIN orders o;
```

Advertencia:

- No usa condición ON.
- Puede generar millones de filas.

Solo debe usarse de forma controlada.

8. JOIN usando USING

8.1. Concepto

Simplifica la sintaxis cuando el nombre de la columna es el mismo.

8.2. Ejemplo

```
SELECT *
FROM users
JOIN orders
USING (id);
```

Condición: La columna debe existir en ambas tablas.

9. JOIN sin Claves Primarias ni Foráneas

9.1. Ejemplo válido

```
SELECT *
FROM users u
JOIN orders o
ON u.username = o.user_id::text;
```

Explicación:

- PostgreSQL no exige PK ni FK.
- Solo evalúa la condición booleana.

Riesgo: inconsistencias y duplicados.

10. Problema de Duplicados

```
-- users
id | username
1  | ana
2  | ana
```

```
SELECT *
FROM users u
JOIN orders o
ON u.username = o.username;
```

Resultado: Una sola fila en orders puede generar múltiples resultados.

11. JOIN vs WHERE (Error común)

11.1. Incorrecto

```
SELECT *
FROM users u, orders o
WHERE u.id = o.user_id;
```

11.2. Correcto

```
SELECT *
FROM users u
JOIN orders o ON u.id = o.user_id;
```

La sintaxis moderna es más clara y evita errores lógicos.

12. JOIN y el Optimizador

12.1. EXPLAIN ANALYZE

```
EXPLAIN ANALYZE
SELECT *
FROM users u
JOIN orders o ON u.id = o.user_id;
```

Impacto de PK/FK:

- Mejores estimaciones de cardinalidad
- Planes más eficientes
- Uso automático de índices

13. Buenas Prácticas

- Usar siempre JOIN explícito
- Definir PK y FK cuando el modelo lo permita
- Evitar JOINs por columnas no únicas
- Usar EXPLAIN ANALYZE para validar rendimiento

14. Resumen Final

- Un JOIN no requiere PK ni FK
- Las claves garantizan integridad, no funcionamiento
- El tipo de JOIN define el comportamiento del resultado
- Un JOIN mal definido puede devolver datos incorrectos

Frase clave:

El JOIN une filas; las claves garantizan que la unión tenga sentido.