**. Conceptos básicos de los JOINS**

Un JOIN combina filas de dos o más tablas basándose en una condición de relación. Esta relación se define típicamente usando una clave primaria (PK) y una clave foránea (FK).

**Sintaxis básica de un JOIN simple (INNER JOIN) conexión CON WHERE:**

SELECT tabla1.columna1, tabla2.columna2, …

FROM tabla1, tabla2 **(CON WHERE AY Q LISTAS TODAS)**

WHERE tabla1.columnaFK = tabla2.columnaPK;

* **Condición de relación**: En WHERE, defines cómo se conectan las tablas (ejemplo: tabla1.columnaFK = tabla2.columnaPK).
* **Producto cartesiano**: Si omites la condición en WHERE, el resultado será todas las combinaciones posibles de las filas entre las tablas involucradas. Esto es casi siempre un error, ya que genera muchos datos irrelevantes.

**2. Tipos de JOINS**

**a) INNER JOIN (Equijoin o JOIN simple)**

Combina filas donde las columnas relacionadas tienen valores iguales.

--SELECT tabla1.columna1(pk), tabla2.columna2(fk)

--FROM tabla1

--INNER JOIN tabla2 ON tabla1.columna\_comun(pk) = tabla2.columna\_comun(fk);

------------------------------------------------------------------------

**where**

SELECT PRODUCTS.NAME,PRODUCTS.PRICE,ORDER\_ITEMS.QUANTITY

FROM PRODUCTS,ORDER\_ITEMS

WHERE PRODUCTS.PRODID=ORDER\_ITEMS.PRODID;

-----------------------------------------------------------------------------------------------

**Where anidado y join =resultado**

SELECT customers.NAME,products.NAME

FROM customers,products,ORDERS,ORDER\_ITEMS

WHERE customers.CUSTID=ORDERS.CUSTID AND ORDERS.ORDID=ORDER\_ITEMS.ORDID AND ORDER\_ITEMS.PRODID=products.PRODID;

SELECT customers.NAME,products.NAME

FROM customers

INNER JOIN **orders ON customers.CUSTID=orders.CUSTID**

---------------------------------------------------------------------------------------------

Sql---------------------------------------------------------------------------------------------

SELECT emp.empno, emp.ename, dept.loc

FROM emp **(ESTAN IMPLICITAMENTE LAS DEMAS AL REALIZAR JOIN)**

JOIN dept ON emp.deptno = dept.deptno;

* **INNER JOIN**: Indica que se va a realizar una unión interna, es decir, solo se devolverán las filas que tienen coincidencias en ambas tablas.
* **ON**: Se usa para definir la condición de unión.
  + En este caso, CUSTOMERS.custid = ORDERS.custid es la condición de unión. Esto significa que el campo custid de la tabla CUSTOMERS debe coincidir con el campo custid de la tabla ORDERS para que se incluya en el resultado de la consulta.

**¿Por Qué es Importante?**

* **Especificación de Relación**: ON permite especificar exactamente cómo se relacionan las filas de diferentes tablas.

---------------------------------------------------------------------------------

**Resumen de Diferencias**

* **Sintaxis Antigua** (con WHERE): Ambas tablas deben ser listadas en la cláusula FROM.
* **Sintaxis Moderna** (con JOIN): La tabla principal se menciona en FROM y las tablas relacionadas se añaden con JOIN.

**Ejemplo Comparativo**

**Sintaxis Antigua:**

sql

SELECT emp.empno, emp.ename, dept.deptno, dept.dname

FROM emp, dept

WHERE emp.deptno = dept.deptno;

**Sintaxis Moderna:**

sql

SELECT emp.empno, emp.ename, dept.dname

FROM emp

INNER JOIN dept ON

------------------------------------------------

**Usar ALIAS para simplificar y mejorar la legibilidad:**

sql

SELECT e.empno, e.ename, d.dname

FROM emp e

INNER JOIN dept d ON e.deptno = d.deptno;

* **Explicación**: Se utilizan alias (e y d) para referirse a las tablas emp y dept, respectivamente, haciendo la consulta más corta y clara.

-----------------------------------------------------

**b) OUTER JOIN**

Incluye filas que no tienen correspondencia en una de las tablas.

* **LEFT OUTER JOIN**: Muestra todas las filas de la tabla izquierda, incluso si no tienen correspondencia en la derecha.
* **RIGHT OUTER JOIN**: Muestra todas las filas de la tabla derecha, incluso si no tienen correspondencia en la izquierda.
* **FULL OUTER JOIN**: Combina ambos tipos, mostrando todas las filas de ambas tablas.
  + **LEFT JOIN:**
  + **Evalúa los valores nulos en la tabla izquierda**.
  + Devuelve **todas las filas de la tabla izquierda** y las filas coincidentes de la tabla derecha.
  + **Si no hay coincidencia en la tabla derecha**, las columnas correspondientes a la tabla derecha se **llenarán con NULL**.
  + Pero **no evalúa NULL en la tabla izquierda**; simplemente muestra todas las filas de la tabla izquierda sin importar si hay valores nulos en las columnas de la izquierda.
  + **2. RIGHT JOIN:**
  + **Evalúa los valores nulos en la tabla derecha**.
  + Devuelve **todas las filas de la tabla derecha** y las filas coincidentes de la tabla izquierda.
  + **Si no hay coincidencia en la tabla izquierda**, las columnas correspondientes a la tabla izquierda se **llenarán con NULL**.
  + **No evalúa NULL en la tabla derecha**; simplemente muestra todas las filas de la tabla derecha sin importar si hay valores nulos en las columnas de la derecha.
  + **3. FULL OUTER JOIN:**
  + **Evalúa los valores nulos en ambas tablas**.
  + Devuelve **todas las filas de ambas tablas**, y **cuando no hay coincidencia** en una de las tablas, **rellena con NULL en las columnas de la tabla sin coincidencia**.
  + **Evalúa los valores nulos de ambas tablas**, es decir, si en la columna de una tabla hay un NULL y no hay coincidencia con la otra tabla, se mostrará un NULL en la columna correspondiente de la otra tabla.

Ejemplo de **LEFT OUTER JOIN**:

sql

Copiar código

SELECT e.name, d.deptno, d.loc

FROM emp e

LEFT JOIN dept d ON e.deptno = d.deptno;

------------------------------------------------------------------------------------------------

Para determinar qué tabla está a la izquierda y cuál a la derecha en una consulta con JOIN, se basa en cómo estructuramos la consulta. Aquí te explico cómo funciona:

**Orden en la Cláusula FROM y JOIN**

* **Tabla de la Izquierda**: Es la primera tabla mencionada en la cláusula FROM.
* **Tabla de la Derecha**: Es la tabla que se menciona después del JOIN.

**Ejemplo con LEFT JOIN**

sql

SELECT SALGRADES.grade, EMPLOYEES.ename

FROM SALGRADES

LEFT JOIN EMPLOYEES ON EMPLOYEES.sal BETWEEN SALGRADES.losal AND SALGRADES.hisal;

En este ejemplo:

* **Tabla de la Izquierda**: SALGRADES (porque es la primera tabla mencionada en la cláusula FROM).
* **Tabla de la Derecha**: EMPLOYEES (porque es la tabla que sigue al LEFT JOIN).

**Uso del Operador + en Sintaxis Antigua**

Cuando usamos el operador + en la cláusula WHERE para simular un LEFT JOIN, lo colocamos en la tabla que está a la derecha del LEFT JOIN en la sintaxis moderna. Esto indica que queremos incluir todas las filas de la tabla de la izquierda incluso si no hay coincidencias en la tabla de la derecha.

**Ejemplo con Sintaxis Antigua**

Para simular la misma consulta con LEFT JOIN utilizando el operador +:

sql

SELECT SALGRADES.grade, EMPLOYEES.ename

FROM SALGRADES, EMPLOYEES

WHERE EMPLOYEES.sal(+) BETWEEN SALGRADES.losal AND SALGRADES.hisal;

En este ejemplo:

* **Tabla de la Izquierda**: SALGRADES (como es la primera tabla mencionada en FROM).
* **Tabla de la Derecha**: EMPLOYEES (la tabla donde se coloca el operador +).

**Concepto Clave**

El operador + se coloca en el lado de la relación donde queremos permitir NULL (es decir, la tabla que puede tener filas sin coincidencias). En la consulta:

* EMPLOYEES.sal(+): Esto permite que las filas de SALGRADES se incluyan incluso si no hay coincidencias en EMPLOYEES.

**Resumen**

* **Tabla de la Izquierda**: Primera tabla mencionada en FROM.
* **Tabla de la Derecha**: Tabla mencionada después del JOIN.
* **Operador** + **en WHERE**: Se coloca en la tabla que puede tener filas sin coincidencias (equivalente a la tabla de la derecha en un LEFT JOIN).

--------------------------------------------------------------------------------------

**c) NON-EQUI JOIN**

Usa operadores diferentes de = para definir la relación entre tablas.

SELECT e.ename, e.sal, s.grade

FROM emp e,salgrade s

JOIN salgrade s ON e.sal BETWEEN s.losal AND s.hisal;

**d) SELF JOIN**

Se usa para comparar filas dentro de la misma tabla. Ejemplo: encontrar empleados y sus jefes.

SELECT trabajador.ename, jefe.ename AS jefe

FROM emp trabajador

JOIN emp jefe ON trabajador.mgr = jefe.empno;

**3. Alias de tablas**

Los alias simplifican el código y mejoran la legibilidad.

* **Uso recomendado**:

sql

Copiar código

SELECT e.ename, d.dname

FROM emp e

JOIN dept d ON e.deptno = d.deptno;

* **Reglas para alias**:
  + Usa nombres cortos pero significativos.
  + Si defines un alias, úsalo en toda la consulta.
  + Los alias solo son válidos dentro de la consulta actual.

**4. Reglas para escribir JOINS**

1. **Especifica condiciones claras de JOIN**:
   * Siempre define al menos n-1 condiciones de JOIN para n tablas, salvo que necesites explícitamente un producto cartesiano.
   * Ejemplo correcto para tres tablas:

sql

Copiar código

SELECT c.name, o.ordid, i.itemid

FROM customer c

JOIN ord o ON c.custid = o.custid

JOIN item i ON o.ordid = i.ordid;

1. **Califica nombres de columnas para evitar ambigüedad**:
   * Si varias tablas tienen columnas con el mismo nombre, califica la columna con el prefijo de la tabla o usa alias.
2. **Combina JOINS con filtros adicionales en la cláusula WHERE**:
   * Ejemplo:

sql

Copiar código

SELECT empno, ename, loc

FROM emp e

JOIN dept d ON e.deptno = d.deptno

WHERE ename = 'KING';

**5. Cosas que NO hacer al usar JOINS**

1. **Omitir las condiciones de JOIN**:
   * Esto genera un **producto cartesiano**, que normalmente no es útil y consume muchos recursos.
   * Ejemplo incorrecto:

sql

SELECT emp.empno, dept.deptno

FROM emp, dept;

1. **Usar OUTER JOIN en ambos lados de la condición**:
   * Incorrecto:

sql

Copiar código

SELECT e.ename, d.deptno

FROM emp e, dept d

WHERE e.deptno (+) = d.deptno (+);

* + Correcto: El operador + solo debe estar en un lado.

1. **Usar operadores no válidos en condiciones de OUTER JOIN**:
   * Evita combinaciones como IN o OR con OUTER JOIN.
2. **Confundir la semántica de los tipos de JOIN**:
   * No uses un INNER JOIN cuando necesitas incluir filas sin coincidencias.

**6. Buenas prácticas**

* **Escribe consultas claras y organizadas**:
  + Usa indentación para distinguir entre las diferentes partes del JOIN y filtros.
* **Prueba la consulta en partes**:
  + Asegúrate de que cada JOIN individual esté funcionando antes de agregar más tablas.
* **Documenta relaciones complicadas**:
  + Para consultas complejas, agrega comentarios que expliquen cada parte.

**7. Resumen visual**

| **Tipo de JOIN** | **¿Qué incluye?** |
| --- | --- |
| INNER JOIN | Solo filas con coincidencias en ambas tablas. |
| LEFT OUTER JOIN | Todas las filas de la tabla izquierda y las coincidentes de la derecha. |
| RIGHT OUTER JOIN | Todas las filas de la tabla derecha y las coincidentes de la izquierda. |
| FULL OUTER JOIN | Todas las filas de ambas tablas, con o sin coincidencias. |
| NON-EQUI JOIN | Relación usando operadores distintos de =. |
| SELF JOIN | Combina una tabla consigo misma. |

**¿Qué es EXISTS?**

La cláusula EXISTS en SQL se utiliza para comprobar si **existe al menos una fila** que cumpla con las condiciones de una subconsulta. No importa cuántas filas se devuelvan, simplemente verifica si hay **una fila** que cumpla con las condiciones especificadas en la subconsulta. Si la subconsulta devuelve **al menos una fila**, la condición se considera **verdadera** y la consulta externa incluirá esa fila en el resultado.

Es especialmente útil cuando no necesitas los datos de la subconsulta, sino solo comprobar su existencia.

**Sintaxis básica de EXISTS:**

SELECT columna1, columna2, ...

FROM tabla1

WHERE EXISTS (

SELECT 1

FROM tabla2

WHERE condiciones

);

* **EXISTS**: Esta palabra clave se usa para comprobar si existe al menos una fila que cumpla las condiciones de la subconsulta.
* **Subconsulta**: Se define entre paréntesis. Puede ser cualquier consulta que retorne filas.
* **SELECT 1**: Aquí usamos 1 porque solo nos interesa saber si hay filas que coincidan, no necesitamos recuperar ninguna columna de la subconsulta. Podríamos usar cualquier valor constante o \*, pero 1 es la forma más eficiente.

**¿Cuándo usar EXISTS?**

**1. Cuando no necesitas los datos de la subconsulta, solo saber si existen filas que cumplan las condiciones.**

* Si solo te interesa saber si una fila cumple con alguna condición (y no necesitas las filas en sí mismas), EXISTS es mucho más eficiente que usar IN o JOIN.

**2. Cuando la subconsulta puede devolver varias filas.**

* Si la subconsulta puede devolver más de una fila, EXISTS es adecuado porque no le importa cuántas filas devuelve la subconsulta, solo que haya al menos una fila que cumpla con las condiciones.

**3. Cuando la subconsulta depende de las filas de la consulta externa (relación correlacionada).**

* Si estás utilizando columnas de la consulta externa en la subconsulta, es un patrón de uso típico de EXISTS. La subconsulta se ejecuta para cada fila de la consulta externa.

**Comparación con otros operadores:**

* **IN**:
  + IN devuelve verdadero si **el valor de la columna externa** se encuentra en la lista de resultados de la subconsulta.
  + Se recomienda usar EXISTS en lugar de IN cuando la subconsulta devuelve muchas filas, ya que IN puede ser menos eficiente.
* **JOIN**:
  + Los JOIN combinan filas de dos tablas basándose en una relación. Si solo te interesa saber si una fila existe en la otra tabla, sin necesidad de unirlas completamente, EXISTS es más adecuado.
* **ANY / SOME**:
  + Estos operadores también pueden usarse con subconsultas, pero **EXISTS** es más eficiente cuando solo necesitas saber si existen filas que cumplen con una condición.

**Ejemplos de uso de EXISTS:**

**1. Comprobación de existencia sin importar el contenido:**

Supón que queremos obtener todos los empleados que tienen un salario mayor que el salario de cualquier empleado en el departamento de ventas:

SELECT E.ENAME, E.SAL

FROM EMP E

WHERE EXISTS (

SELECT 1

FROM EMP E1

WHERE E1.DEPTNO = E.DEPTNO

AND E1.SAL < E.SAL

);

* En este caso, la subconsulta verifica si existe **al menos un empleado** en el mismo departamento cuyo salario sea **menor** al salario del empleado actual.
* **EXISTS** devuelve **TRUE** si se encuentra cualquier empleado con ese salario menor, independientemente de cuántos empleados haya.

**2. Usando EXISTS con una subconsulta correlacionada:**

Supón que queremos obtener los nombres de los empleados que trabajan en departamentos con más de 3 empleados:

SELECT E.ENAME

FROM EMP E

WHERE EXISTS (

SELECT 1

FROM EMP E1

WHERE E1.DEPTNO = E.DEPTNO

GROUP BY E1.DEPTNO

HAVING COUNT(\*) > 3

);

* En este caso, la subconsulta correlacionada está verificando que el departamento del empleado (E.DEPTNO) tenga más de tres empleados.
* **EXISTS** devuelve **TRUE** solo si el departamento del empleado tiene más de tres empleados.

**Patrones de uso recomendados de EXISTS:**

1. **Cuando necesitas verificar la existencia de una relación, no los datos:**
   * Usar EXISTS para verificar si hay filas que cumplan una condición sin necesidad de devolver esas filas.
2. **Cuando la subconsulta involucra la tabla externa**:
   * Si la subconsulta hace referencia a las columnas de la consulta externa (es una **subconsulta correlacionada**), EXISTS es más adecuado.
3. **Cuando la subconsulta puede devolver múltiples filas**:
   * Si la subconsulta puede devolver varias filas, **EXISTS** será más eficiente que IN, ya que solo verifica la existencia de las filas, sin importar cuántas se devuelvan.

**Resumen de la sintaxis de EXISTS:**

SELECT columnas

FROM tabla\_externa

WHERE EXISTS (

SELECT 1

FROM tabla\_interna

WHERE condiciones

);

* **SELECT 1**: Usamos 1 porque solo nos interesa saber si hay al menos una fila que cumpla con las condiciones, no necesitamos recuperar datos.
* **Subconsulta correlacionada**: Si usas columnas de la tabla externa en la subconsulta, asegúrate de hacer referencia a ellas correctamente (esto es lo que hace que la subconsulta sea "correlacionada").

¡Espero que ahora te quede claro cómo funciona EXISTS y cuándo es útil! Si tienes más preguntas o ejemplos específicos en los que te gustaría usar EXISTS, no dudes en preguntar. 😊

**Qué es una subconsulta correlacionada?**

Una **subconsulta correlacionada** es una subconsulta que hace referencia a **columnas de la consulta externa**. Es decir, en la subconsulta se utilizan valores de la fila actual de la consulta externa, y la subconsulta se evalúa **una vez por cada fila de la consulta principal**.

**Sintaxis de una subconsulta correlacionada:**

sql

CopiarEditar

SELECT columnas

FROM tabla\_externa E

WHERE EXISTS (

SELECT 1

FROM tabla\_interna E1

WHERE E1.columna = E.columna

-- Otras condiciones

);

**Explicación:**

* **Consulta externa (principal)**: En la consulta externa, se refiere a una tabla (por ejemplo, tabla\_externa E) y en su WHERE o en otro lugar, se hace referencia a columnas de esa tabla.
* **Subconsulta correlacionada**: En la subconsulta, se hace referencia a la columna de la tabla externa. **E1.columna = E.columna** es la correlación que conecta la subconsulta con la fila actual de la consulta externa.

//////////////////////////////////////////////////////////////////////////////////////////////

**Unión de tablas (JOINS)**

Cuando necesitas datos de varias tablas relacionadas, debes usar una unión o **JOIN**. Esto se basa en columnas con valores comunes, típicamente **claves primarias (PK)** y **claves foráneas (FK)**.

**Sintaxis básica de JOIN:**

sql

Copiar código

SELECT tabla1.columna1, tabla2.columna2, ...

FROM tabla1, tabla2

WHERE tabla1.columnaFK = tabla2.columnaPK;

**Ejemplo inicial:**

Tienes dos tablas:

1. **EMP** (empleados):
   * **empno** (clave primaria)
   * **ename** (nombre)
   * **deptno** (clave foránea al departamento)
2. **DEPT** (departamentos):
   * **deptno** (clave primaria)
   * **dname** (nombre del departamento)

**Consulta para combinar ambas tablas y mostrar el nombre del empleado, su número y el nombre del departamento:**

sql

Copiar código

SELECT emp.ename, emp.empno, dept.dname

FROM emp, dept

WHERE emp.deptno = dept.deptno;

**Reglas importantes:**

1. **Número de condiciones de JOIN**: Para unir N tablas necesitas al menos (N - 1) condiciones. Por ejemplo:
   * 2 tablas → 1 condición.
   * 4 tablas → 3 condiciones.
2. **Evitar el producto cartesiano**: Si omites la condición del WHERE, SQL combinará **todas las filas de ambas tablas**, resultando en combinaciones inútiles.

**Alias de tablas**

Usar alias simplifica el código y mejora la claridad.

**Ejemplo:**

sql

Copiar código

SELECT e.ename, e.empno, d.dname

FROM emp e, dept d

WHERE e.deptno = d.deptno;

**Reglas para los alias:**

* **Significativos:** Usa alias cortos pero descriptivos (como e para empleados).
* Si defines un alias en FROM, úsalo en toda la consulta.
* Los alias sólo aplican en la consulta donde se definen.

**Tipos de JOINS**

**1. INNER JOIN (EQUIJOIN):**

Este es el JOIN básico que hemos visto. Muestra solo las filas donde hay coincidencias entre las tablas.

**Ejemplo:**

sql

Copiar código

SELECT e.ename, d.dname

FROM emp e

INNER JOIN dept d

ON e.deptno = d.deptno;

**Equivalente a:**

sql

Copiar código

SELECT e.ename, d.dname

FROM emp e, dept d

WHERE e.deptno = d.deptno;

**2. NON-EQUIJOIN:**

Se usa cuando las tablas se relacionan por una condición diferente al igual (=).

**Ejemplo**: Relacionar empleados con un rango salarial en una tabla de categorías salariales (**SALGRADE**).

sql

Copiar código

SELECT e.ename, e.sal, s.grade

FROM emp e, salgrade s

WHERE e.sal BETWEEN s.losal AND s.hisal;

**3. OUTER JOIN:**

Incluye filas de una tabla aunque no tengan coincidencias en la otra.

* **LEFT OUTER JOIN**: Incluye todas las filas de la tabla izquierda, agregando NULL en las columnas faltantes de la derecha.

sql

Copiar código

SELECT e.ename, d.dname

FROM emp e

LEFT OUTER JOIN dept d

ON e.deptno = d.deptno;

* **RIGHT OUTER JOIN**: Lo contrario del anterior: incluye todas las filas de la tabla derecha.
* **FULL OUTER JOIN**: Combina ambos. Muestra todas las filas, llenando con NULL donde no haya coincidencias.

**Sintaxis alternativa con + (propietario de Oracle):**

sql

Copiar código

SELECT e.ename, d.dname

FROM emp e, dept d

WHERE e.deptno(+) = d.deptno;

**Restricciones para el operador +:**

* Solo puede estar en un lado de la condición.
* No se puede usar con IN ni combinar con OR.

**Relaciones reflexivas**

Un JOIN en la misma tabla ocurre cuando una fila se relaciona consigo misma.

**Ejemplo:**

Relación jefe-empleado:

sql

Copiar código

SELECT trabajador.ename || ' trabaja para ' || jefe.ename AS jerarquia

FROM emp trabajador, emp jefe

WHERE trabajador.mgr = jefe.empno;

**Errores comunes y cómo evitarlos**

1. **Olvidar condiciones en WHERE**: Genera un producto cartesiano. Siempre especifica las relaciones entre tablas.
2. **Ambigüedad en columnas**: Si dos tablas tienen columnas con el mismo nombre, califícalas con el nombre de la tabla o alias.
3. **Alias inconsistentes**: Si usas un alias, úsalo en toda la consulta.
4. **JOIN mal estructurado**: Revisa que las condiciones de JOIN sean correctas y completas.

**Resumen de reglas clave:**

1. Usa alias para facilitar la lectura.
2. Verifica las condiciones de JOIN para evitar productos cartesianos.
3. Califica columnas ambiguas con el nombre de la tabla o alias.
4. El número de condiciones de JOIN debe ser al menos (N - 1) para N tablas.
5. Usa OUTER JOIN si quieres incluir filas que no tienen coincidencias.

¿Te gustaría practicar con un ejemplo personalizado o necesitas ayuda con algo específico? 😊

**Identificar la tabla izquierda y derecha**

En un LEFT JOIN:

* La **tabla de la izquierda** es la que aparece **antes del LEFT JOIN** en la consulta.
* La **tabla de la derecha** es la que aparece **después del LEFT JOIN**.

En tu consulta:

sql

Copiar código

SELECT P.NAME, SUM((P.PRICE\*OT.QUANTITY)) AS TOTAL\_VENTAS\_PRODUCTO

FROM PRODUCTS P

LEFT JOIN ORDER\_ITEMS OT ON P.PRODID = OT.PRODID

GROUP BY P.NAME;

* PRODUCTS (P) es la tabla **de la izquierda**.
* ORDER\_ITEMS (OT) es la tabla **de la derecha**.

**¿Qué hace el LEFT JOIN?**

1. Un LEFT JOIN garantiza que:
   * **Todas las filas de la tabla izquierda (PRODUCTS) estarán en el resultado**.
   * Si no hay coincidencias en la tabla derecha (ORDER\_ITEMS), los valores de las columnas de la derecha serán NULL.
2. En este caso:
   * Cada producto de PRODUCTS estará presente en el resultado, incluso si no tiene registros relacionados en ORDER\_ITEMS.

**Cómo evitar la confusión**

Si no estás seguro de cuál es la tabla izquierda o derecha:

* Revisa el orden en que aparecen en la consulta.
* Recuerda: **la tabla mencionada primero en el FROM es la izquierda**.

En tu ejemplo, si quisieras hacer lo contrario (mostrar todos los datos de ORDER\_ITEMS y, si no tienen un producto asociado, incluir NULL para las columnas de PRODUCTS), usarías un RIGHT JOIN o invertirías el orden de las tablas.