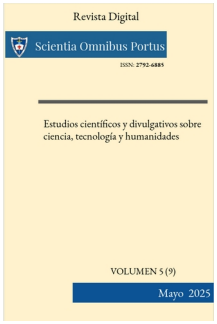
SciOP. 2024, 5(9)

Scientia Omnibus Portus
ISSN: 2792-6885

https://iescelia.org/ojs



Artículo

Extracción de Datos de Sistemas Gestores de Bases de Datos

Eduardo Tapia
etappad369@g.educaand.es

RESUMEN

Las instrucciones JOIN en SQL se utilizan para vincular filas de dos o más tablas basadas en una relación establecida entre ellas. Esto permite recuperar datos de múltiples fuentes de forma cohesionada.

Las consultas SQL funcionan como un medio indispensable para interactuar con las bases de datos, permitiendo recuperar, modificar o eliminar información almacenada de forma rápida y sencilla. Son una herramienta imprescindible a la hora de administrar y manipular los datos almacenados en una base de datos relacional, pudiendo extraer, filtrar y ordenar registros de manera flexible y precisa.

El lenguaje SQL resulta vital para llevar a cabo análisis y gestión efectiva de los datos en cualquier sistema que emplee bases de datos relacionales, ya que gracias a sus sentencias se puede acceder, transformar e inspeccionar la información guardada de una forma potente pero al mismo tiempo intuitiva. Su sintaxis normalizada facilita enormemente las consultas a la base de datos independientemente del motor que se esté utilizando.

Cada tipo de JOIN es apropiado para distintos escenarios dependiendo de la necesidad de extraer y relacionar información entre las fuentes.

PALABRAS CLAVE : SQL, Bases de Datos, Modelo Entidad Relación, Oracle , PostgreSQL.-

INTRODUCCIÓN

En SQL, los JOINs se utilizan para juntar filas de dos tablas o más según un campo común entre ellas, por lo tanto, devolverá datos de varias tablas. Es decir, un JOIN ocurre cada vez que dos o más tablas se unen en una consulta SQL.

Hay más tipos de JOINs en SQL que los explicados aquí, como CROSS JOIN, O SELF JOIN, pero no todos estos JOINs son compatibles con todos los sistemas de bases de datos. Los tipos más importantes son:

INNER JOIN: devuelve todas las filas cuando al menos hay una coincidencia en ambas tablas.

LEFT JOIN: devuelve todas las filas de la tabla de la izquierda y las filas coincidentes de la tabla de la derecha.

RIGHT JOIN: devuelve todas las filas de la tabla de la derecha y las filas coincidentes de la tabla de la izquierda.

OUTER JOIN: devuelve todas las filas en ambas tablas, a la izquierda y a la derecha. También conocido como FULL OUTER JOIN.

INNER JOIN

INNER JOIN es una sentencia utilizada en SQL para combinar filas de dos o más tablas basadas en una condición de unión. Esta condición de unión especifica cómo las filas de una tabla se relacionan con las filas de otra tabla para formar el resultado conjunto.

INNER JOIN se utiliza cuando solo desea recuperar registros que tengan coincidencias en ambas tablas (Figura 1). En otras palabras, solo desea obtener los registros que cumplan con la condición de unión especificada.

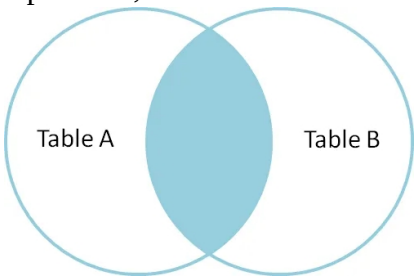


Figura 1. Representación de inner join en teoría de conjuntos

La sintaxis fundamental de INNER JOIN involucra enumerar las tablas a conectar después de FROM, seguido de INNER JOIN y la tabla a asociar. La condición de enlace se especifica utilizando la palabra clave ON, donde se establece la relación entre los campos de las tablas. La INNER JOIN solamente combina filas cuando existe coincidencia en los valores vinculados, lo que permite extraer datos de múltiples tablas bases de datos relacionadas. Este tipo de consulta SQL es

muy útil para recuperar datos cruzados que cumplen condiciones específicas. Al igual que otras sentencias, INNER JOIN también permite realizar operaciones de selección, proyección, agregación y ordenamiento sobre los resultados.

```
SELECT columnas

FROM tabla1
INNER JOIN tabla2
ON tabla1.columna = tabla2.columna;
```

EJEMPLO

En este ejemplo se demuestra el uso de INNER JOIN para conectar dos tablas relacionadas entre sí. La tabla "empleado" contiene la información del personal de la empresa, mientras que la tabla "departamento" almacena los datos de las diferentes áreas y secciones. Para recuperar los nombres tanto de los trabajadores como de sus respectivos departamentos, es necesario utilizar un JOIN que vincule ambas tablas a través de la columna "id_departamento". De este modo, podremos obtener una lista completa con los empleados y los departamentos a los cuales se encuentran adscritos. El JOIN INNER eliminará cualquier registro que no posea una coincidencia en la otra tabla, mostrando solamente los resultados donde exista una asociación directa entre los campos relacionados.

Empleado (id_empleado, nombre, id_departamento)

Departamento(id_departamento, nombre)

De esta forma, lograremos conocer el personal que integra cada uno de los departamentos gracias a la consulta generada por esta operación de unión.

```
SELECT empleado.nombre , departamento.nombre
FROM empleado
INNER JOIN departamento
ON empleado.id_departamento = departamento.id_departamento;
```

Dentro de este ejemplo, la declaración INNER JOIN combina las filas de las tablas "empleado" y "departamento" donde el valor de "id_departamento" en la tabla "empleado" coincide con el valor de "id_departamento" en la tabla "departamento".

Cuando se trabaja con múltiples tablas en una consulta SQL, el uso de alias en varias uniones hace que las consultas SQL sean más legibles, claras y comprensibles. Esto ayuda a los desarrolladores a escribir consultas más eficientes y a evitar errores al trabajar con bases de datos complejas.

Supongamos que tenemos tres tablas: "clientes", "pedidos" y "productos". Queremos obtener información sobre los pedidos realizados por cada cliente y los productos asociados a cada pedido. Utilizaremos alias para las tablas y campos para hacer que la consulta sea más entendible.

```
SELECT c.nombre AS nombre_cliente, p.numero_pedido, pr.nombre AS nombre_producto
FROM clientes AS c
INNER JOIN pedidos AS p
ON c.id_cliente = p.id_cliente
INNER JOIN productos AS pr
ON p.id_producto = pr.id_producto;
```

En este ejemplo, "c" es el alias de la tabla "clientes", "p" representa la tabla "pedidos" y "pr" es el alias para la tabla "productos". Se utilizan estos apodos para hacer referencia a las tablas en la consulta de una manera más clara y concisa.

Asimismo, se emplean alias para los campos específicos como "nombre_cliente" para el nombre del cliente, "número_pedido" para el número del pedido realizado y "nombre_producto" como alias del nombre del producto. El uso de estas abreviaturas hace que la consulta sea más legible y comprensible al mismo tiempo que simplifica la escritura de los nombres completos de las tablas y campos en cada una de las líneas de código SQL.

La cláusula WHERE se utiliza para filtrar filas basadas en parámetros específicos. Podemos implementar condiciones tanto en las columnas de las tablas unidas como en cualquier campo presente en las mismas. Supongamos que con nuestras dos bases de datos, "empleado" y "departamento", y queremos obtener los nombres de los trabajadores que pertenecen al área de "Ventas". Asimismo, es posible requerir datos de los ejecutivos encargados de la división comercial cuyo rendimiento fue sobresaliente en años anteriores. De esta forma, filtraríamos registros atendiendo a múltiples variables de manera simultánea.

```
SELECT empleado.nombre AS nombre_empleado
FROM empleado
INNER JOIN departamento ON empleado.id_departamento = departamento.id_departamento
WHERE departamento.nombre = 'Ventas';
```

Este consulta selecciona el nombre de los empleados que pertenecen al departamento de ‘Ventas’. Luego de aplicar INNER JOIN y WHERE, la tabla de resultados contendría únicamente a los empleados cuyo departamento es ‘Ventas’.

La cláusula ORDER BY utiliza para ordenar los resultados en función de una o más columnas, ya sea de manera ascendente (ASC) o descendente (DESC). Podemos aplicar ORDER BY a columnas de cualquiera de las tablas unidas. Supongamos que queremos obtener una lista de empleados ordenada alfabéticamente por su nombre e igualmente deseamos incluir el nombre de su departamento.

```
SELECT empleado.nombre AS nombre_empleado, departamento.nombre AS
nombre_departamento
FROM empleado
INNER JOIN departamento ON empleado.id_departamento = departamento.id_departamento
ORDER BY empleado.nombre ASC;
```

Este párrafo describe una consulta que selecciona el nombre de los empleados y el nombre de su respectivo departamento, ordenados alfabéticamente por el apellido del funcionario de manera ascendente.

La cláusula GROUP BY utiliza para congregar filas que posean un mismo valor en una o más columnas y aplicar funciones de agregación a los grupos resultantes. Podemos emplear GROUP BY en columnas de cualquiera de las tablas unidas. Supongamos que deseamos contar cuántos empleados hay en cada uno de los departamentos y también queremos exhibir el nombre de cada área.

```
SELECT departamento.nombre AS nombre_departamento, COUNT(empleado.id_empleado) AS
total_empleados
FROM empleado
INNER JOIN departamento ON empleado.id_departamento = departamento.id_departamento
GROUP BY departamento.nombre;
```

LEFT JOIN

Un LEFT JOIN selecciona todas las filas de la tabla de la izquierda y coincide las filas de la derecha cuando exista correspondencia. De no hallarse emparejamiento, la consulta rellenará con VALORES NULOS los campos de la tabla de la derecha.

Sintaxis básica :

```
SELECT columnas
FROM tabla1
LEFT JOIN tabla2
ON tabla1.columna_relacion = tabla2.columna_relacion;
```

- tabla1: es la **tabla izquierda** (la principal).
- tabla2: es la **tabla derecha** (la complementaria).
- La condición ON indica cómo se relacionan las tablas.

Ejemplo práctico

Imagina estas dos tablas:

Tabla: empleados

id	nombre	id_departamento
1	Ana	10
2	Luis	20
3	María	NULL
4	Jorge	30

Tabla: departamentos

id	nombre
10	Contabilidad
20	Ventas

Realizamos la siguiente consulta :

```
SELECT empleados.nombre, departamentos.nombre AS departamento
FROM empleados
LEFT JOIN departamentos
ON empleados.id_departamento = departamentos.id;
```

Nos daría de resultado :

nombre	departamento
Ana	Contabilidad
Luis	Ventas
María	NULL
Jorge	NULL

¿Por qué Jorge y María tienen NULL?

María no tiene asignado ningún departamento (id_departamento es NULL).

Jorge tiene id_departamento = 30, pero ese departamento no existe en la tabla departamentos.

¿Cuándo usar LEFT JOIN?

Cuando se desea visualizar la totalidad de los datos de una tabla principal, a pesar de que estos no guarden relación directa con la secundaria. Esto resulta muy útil para detectar información ausente, como empleados sin asignar, productos cuyas ventas no han sido registradas, alumnos que no cuentan con matrícula u otros registros similares.

RIGHT JOIN

¿Qué hace el RIGHT JOIN ?

Un RIGHT JOIN devuelve:
Todos los registros de la tabla de la derecha.
Las coincidencias de la tabla de la izquierda.
Si no existe una coincidencia, los valores de la tabla izquierda serán NULOS.

La diferencia clave con el LEFT JOIN :

El LEFT JOIN : devuelve todos los registros de la tabla izquierda, aunque no exista una coincidencia en la derecha.

El RIGHT JOIN : devuelve todos los registros de la tabla derecha, aunque no exista una coincidencia en la izquierda.

SINTAXIS

```
SELECT columnas
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.columna_relacion = tabla2.columna_relacion;
```

---

tabla1: es la **izquierda**.

tabla2: es la **derecha** (y es la que se mantiene completa en el resultado).

EJEMPLO

id	nombre	id_departamento
1	Ana	10
2	Luis	20
3	Carlos	NULL

id	nombre
10	Contabilidad
20	Ventas
30	Recursos Humanos

Realizamos la siguiente consulta :

```
SELECT empleados.nombre, departamentos.nombre AS departamento
FROM empleados
RIGHT JOIN departamentos
ON empleados.id_departamento = departamentos.id;
```

Resultado :

nombre	departamento
Ana	Contabilidad
Luis	Ventas
NULL	Recursos Humanos

EXPLICACIÓN

Ana y Luis aparecen porque sus id_departamento coinciden con los id de los departamentos.

Recursos Humanos aparece aunque no tiene empleados asignados, y por eso su nombre es NULL.

USO ADECUADO DE RIGHT JOIN

Cuando quieras ver toda la información de la tabla de la derecha, y no desees perder datos aunque no tengan coincidencia en la izquierda.

Es útil para responder cosas como:

➤ ¿Qué departamentos no tienen empleados?
➤ ¿Qué productos no han sido comprados?
➤ ¿Qué cursos no tienen alumnos inscritos?

Si tienes dificultad para entender la diferencia entre JOIN IZQUIERDO y JOIN DERECHO, simplemente invierte el orden de las tablas y usa JOIN IZQUIERDO.

OUTER JOIN

Las consultas externas son un tipo de instrucción en SQL que posibilita combinar filas de dos tablas, incorporando aquellas que no presentan coincidencias mutuas.

Se descompone en tres variantes distintas:

LEFT OUTER JOIN → (o simplemente JOIN IZQUIERDO) que incluye todas las filas de la primera tabla especificada y las filas coincidentes de la segunda tabla.

RIGHT OUTER JOIN → (o únicamente JOIN DERECHO) que incluye todas las filas de la segunda tabla especificada y las filas coincidentes de la primera tabla.

FULL OUTER JOIN → (que combina ambas opciones anteriores) recupera todas las filas de ambas tablas especificadas, haya o no coincidencias entre ellas.

LEFT OUTER JOIN

Devuelve todas las filas de la tabla izquierda y los datos correspondientes de la tabla derecha.

Si no existe una coincidencia, la sección derecha se completará con NULL.

```
SELECT empleados.nombre, departamentos.nombre AS departamento
FROM empleados
LEFT OUTER JOIN departamentos
ON empleados.id_departamento = departamentos.id;
```

Esto muestra a todos los empleados, incluso si no tienen un departamento asignado.

RIGHT OUTER JOIN

Devuelve todas las filas de la tabla derecha y los datos vinculados de la izquierda.

Si no hay una coincidencia, la parte izquierda será NULL.

```
SELECT empleados.nombre, departamentos.nombre AS departamento
FROM empleados
RIGHT OUTER JOIN departamentos
ON empleados.id_departamento = departamentos.id;
```

Esto muestra todos los departamentos, incluso aquellos que no tienen empleados asignados actualmente.

FULL OUTER JOIN

El JOIN FULL permite combinar dos tablas para devolver todas las filas, incluyendo aquellas que no tienen coincidencia en la otra tabla. De esta forma, entrega una visión integral de los datos en ambas vistas, sin omitir ninguna entrada.

Devuelve tanto los registros que coinciden en ambas tablas como aquellos que son únicos de cada una. Si no hay emparejamiento, rellena los campos faltantes con valores nulos.

id	nombre	id_departamento
1	Ana	10
2	Luis	20
3	Carla	NULL

id	nombre
10	Contabilidad
20	Ventas
30	Recursos Humanos

```
SELECT empleados.nombre AS empleado, departamentos.nombre AS departamento
FROM empleados
FULL OUTER JOIN departamentos
ON empleados.id_departamento = departamentos.id;
```

empleado	departamento
Ana	Contabilidad
Luis	Ventas
Carla	NULL
NULL	Recursos Humanos

Esto resulta útil para comparar listados y detectar discrepancias, como por ejemplo productos registrados contra productos vendidos.

Esta función también resulta valiosa para auditorías y revisiones donde se requiera confrontar información incompleta proveniente de distintas fuentes, asegurando que no se pierda ningún elemento de ambos conjuntos.

DIFERENCIA ENTRE EL USO DEL PRODUCTO CARTESIANO Y EL USO DE JOIN EN CONSULTAS SQL.-

En SQL, el producto cartesiano y las fusiones se utilizan para combinarse con filas en dos o más tablas. Sin embargo, tienen un propósito y funcionamiento diferentes. Las siguientes son las diferencias sustanciales entre las dos.

En primer lugar, el producto cartesiano es una operación que devuelve todas las combinaciones posibles de filas entre dos tablas. Es decir, cada fila en la primera tabla se combina una vez con todas las filas de la segunda tabla.

Por otro lado, JOIN se utiliza para combinar filas entre dos mesas basadas en una condición lógica. Es decir, la combinación se realiza en función de la igualdad de valores, generalmente relacionados con una clave foránea y una clave primaria. También hay varios tipos de JOIN que se pueden utilizar en SQL, como INNER JOIN, LEFT JOIN, RIGHT JOIN entre los principales utilizados.

Diferencias :

Característica	Producto Cartesiano	JOIN
Relación entre tablas	No requiere relación	Basado en condición lógica (ON)
Número de filas	Multiplica el total de filas de ambas tablas	Solo filas con coincidencias (dependiendo del tipo de JOIN)
Uso típico	Para pruebas o combinaciones completas	Para consultar datos relacionados lógicamente
Eficiencia	Poco eficiente en tablas grandes	Más eficiente y controlado
Resultado	Todas las combinaciones posibles	Filas relacionadas con sentido lógico

Sentencia EXPLAIN en MySQL

La sentencia EXPLAIN en MySQL es una herramienta poderosa para analizar y optimizar consultas. Explica cómo el optimizador de MySQL espera ejecutar una consulta, proporcionando detalles sobre el orden de lectura de tablas, los índices utilizados, el tipo de unión y muchos otros puntos clave.

La tabla que se obtiene con EXPLAIN tiene columnas destinadas a ayudar a comprender cómo se desglosará la consulta.

Algunas de las columnas más importantes en este sentido son las siguientes:

- id, usado como un identificador para indicar el paso de la consulta;
- select_type, utilizado como un nombre para el tipo de consulta en ese paso;
- table, representando el nombre de la tabla a la que se accede en este paso, etc.

Sin embargo, la descripción de las columnas de la tabla no puede concluirse sin mencionar las siguientes:

- type, que simbólicamente puede representar cualquier tipo de unión ;
- possible_keys, identifica los posibles índices que se pueden usar para encontrar las filas ;
- key, para el índice real corto o una clave única que se usará para acceder a la tabla ;
- key_len, informando sobre la longitud del índice que se utilizará ;
- ref, puede indicar la columna comparado con algún tipo de constante o columna ;
- rows, una estimación del número de filas que serán examinadas ; y
- Extra, que normalmente incluye algunas explicaciones en mysql, como Using where , Using index , etc..

Sintaxis Básica

La sintaxis general es:

EXPLAIN SELECT * FROM nombre_tabla WHERE condición;

Debes usar EXPLAIN si:

* Su consulta es lenta;

* Necesitas asegurarte de que un ejecutable está utilizando índices;

* Implementas calidad de base de datos y optimizas la estructura de tus tablas y consultas de datos;

* Deseas entender mejor cómo MySQL procesa el SQL.

* Usa EXPLAIN antes de ajustar cualquier experiencia o consulta;

* Úsalo junto con EXPLAIN ANALYZE en versiones recientes de MySQL para obtener más métricas detalladas;

* Y no olvides usar los índices correctamente; evita los ‘tipos’ de ‘ALL’ donde sea posible ya que esto implica un escaneo completo.

Registro de consultas lento

El Slow Query Log de MySQL es un archivo de log que ayuda a diagnosticar consultas que demoran mucho tiempo en finalizar. Es esencial para detectar cuellos de botella y optimizar el rendimiento del servidor.

¿Qué es el Slow Query Log como tal?

Es un archivo de log con las consultas SQL que han excedido el límite establecido de tiempo de ejecución. Este archivo permite que analice cuál o cuáles sentencias están dañando el rendimiento de su sistema en general. \ Ajusta el long_query_time para maximizar la distancia entre la señal (consultas problemáticas) y el ruido (todas las demás).

Mantén el log con un tamaño razonable (por ejemplo, mediante rotación de logs).

Usa EXPLAIN para averiguar por qué las consultas corren lentas. Implementa índices en sus tablas que los usuarios típicamente consultan.

En resumen, el Slow Query Log es una parte fundamental del monitoreo MySQL. Permite identificar consultas problemáticas e ineficientes para cada una de estas, luego optimizarlas, permitiendo un rendimiento óptimo de todo el sistema.

Puedes habilitarlo desde el archivo de configuración `my.cnf` o dinámicamente desde el cliente MySQL:

SET GLOBAL slow_query_log = 'ON';

SET GLOBAL slow_query_log_file = '/ruta/a/slow-query.log';

SET GLOBAL long_query_time = 1;

También puedes establecerlo en el archivo `my.cnf` de la siguiente forma:

[mysqld]

slow_query_log = 1

slow_query_log_file = /var/log/mysql/slow-query.log

long_query_time = 1

log_queries_not_using_indexes = 1

Parámetros Importantes

- slow_query_log: habilita o deshabilita el registro.

- slow_query_log_file: ruta del archivo de log.

- long_query_time: tiempo mínimo (en segundos) que una consulta debe tardar para ser registrada.

- log_queries_not_using_indexes: registra también consultas que no usan índices (aunque sean rápidas).

Para revisar el log puedes abrir el archivo directamente o usar herramientas como `mysqldumpslow` o `pt-query-digest` para resumir las consultas más frecuentes y costosas:

mysqldumpslow /var/log/mysql/slow-query.log

pt-query-digest /var/log/mysql/slow-query.log

Ajusta el `long_query_time` para encontrar el equilibrio entre ruido y efectividad.

Asegúrate de que el log no crezca indefinidamente (usa rotación de logs).

Usa EXPLAIN para entender por qué una consulta es lenta.Asegúrate de tener índices adecuados en tus tablas.

CONCLUSIONES

Obtener datos específicos de diferentes tablas y presentar la información de manera más efectiva puede ser útil para la toma de decisiones. Cuando trabajamos con bases de datos relacionales, una tabla fuerte representa una entidad independiente mientras que una tabla débil representa una entidad que depende de otra. Estas relaciones, que son las dependencias entre tablas, consisten en columnas que comparten el mismo tipo de datos. Por lo tanto, la cláusula JOIN nos permite asociar dos o más tablas en función de una columna que tengan en común, lo que facilita el análisis y comprensión de la información al vincular y agrupar los datos de manera relacional.

REFERENCIAS BIBLIOGRÁFICAS :

W3Schools SQL Tutorial – https://www.w3schools.com/sql/

PostgreSQL Documentation – https://www.postgresql.org/docs/

REFERENCIAS BIBLIOGRÁFICAS RECOMENDADAS :

Bases de Datos.-30 mayo 2012

de Luis Hueso Ibáñez Galindo (Autor)
Editorial: RA-MA S.A. Editorial y Publicaciones

ISBN-13 : 978-8499641577

Aprende SQL en un fin de semana: El curso definitivo para crear y consultar bases de datos

de Antonio Padial Solier (Autor)
ISBN-13 : 978-1520363462

Eduardo Tapia : Profesor de Secundaria .-