Lab 2 - Operaciones JOIN y Consultas Anidadas en SQL

David Moreno Lumbreras & Daniela Patricia Feversani
GSyC, EIF. URJC.

Laboratorio de Bases de Datos (BBDD)

Curso 24-25

Contenidos

L2.1 Operaciones JOIN

Operaciones JOIN

- Las operaciones **JOIN** se utilizan para combinar datos de dos o más tablas.
- **Combinación de Datos**: Las operaciones JOIN combinan información de diferentes tablas para obtener datos más completos y relevantes.
- **Tablas Relacionadas**: JOIN conecta tablas basadas en claves primarias y foráneas, facilitando la obtención de información coherente.
- **Concepto de Conjuntos**: JOIN se basa en el concepto matemático de conjuntos, combinando filas de tablas según criterios.
- **Eficiencia y Optimización**: JOIN puede ser costoso en términos de rendimiento, por lo que se optimiza para mejorar la eficiencia en consultas.
- La estructura general de una operación JOIN es la siguiente:

```
SELECT columns_from_both_tables
FROM table1
[INNER, LEFT, RIGHT, FULL] JOIN table2
ON table1.column1 = table2.column2
```

Operaciones JOIN

- **INNER JOIN**: Coincidencias entre A y B. Devuelve solo los registros que coinciden en ambas tablas.

```sql
-- Obtener información sobre los clientes que han realizado pedidos
SELECT customer.customer_id, customer.fname, customer.lname, orderinfo.date_placed
FROM customer
INNER JOIN orderinfo ON customer.customer_id = orderinfo.customer_id;

-- Mostrar el nombre y apellido de cada alumno junto con las asignaturas
--en las que está matriculado.
SELECT estudiantes.nombre, estudiantes.apellido, asignaturas.nombre
FROM estudiantes
INNER join matriculas on estudiantes.matricula = matriculas.matricula
INNER JOIN asignaturas ON matriculas.idasignatura = asignaturas.idasignatura;
```

Operaciones JOIN

- **LEFT JOIN**: Todos los de A cruzados con B (NULL si no coinciden). Devuelve todos los registros de la tabla izquierda, incluso si no hay coincidencias en la tabla derecha.

```sql
-- Muestra todos los productos junto con su cantidad disponible en stock,
-- incluyendo productos sin stock.
SELECT item.item_id, item.description, stock.quantity
FROM item
LEFT JOIN stock ON item.item_id = stock.item_id;

-- Muestra nombre, créditos y semestre de una asignatura junto con el nombre
-- del profesor y departamento que la imparte.
SELECT asignaturas.nombre, asignaturas.creditos, asignaturas.semestre,
profesores.nombre, profesores.departamento
FROM asignaturas
LEFT JOIN profesores ON asignaturas.idprofesor = profesores.idprofesor;
```

Operaciones JOIN

- `RIGHT JOIN`: Todos los de B cruzados con A (`NULL` si no coinciden). Devuelve todos los registros de la tabla derecha, incluso si no hay coincidencias en la tabla izquierda.

```sql
-- Órdenes de compra junto con la información del cliente asociado.
SELECT customer.fname, customer.lname, orderinfo.date_placed
FROM customer
RIGHT JOIN orderinfo ON customer.customer_id = orderinfo.customer_id;

-- Muestra todos los productos junto con su cantidad disponible en stock,
-- no incluyendo productos sin stock.
SELECT item.item_id, item.description, stock.quantity
FROM item
RIGHT JOIN stock ON item.item_id = stock.item_id;
```

Operaciones JOIN

- `FULL JOIN`: Todos los de A y B sin importar si coinciden o no (`NULL` si no coinciden).

```
SELECT item.item_id, description, cost_price, sell_price, barcode_ean
FROM item
FULL JOIN barcode
ON item.item_id = barcode.item_id;
```

- Si añadimos una cláusula `WHERE` podemos implementar operaciones de sustracción de conjuntos.

```
-- Selecciona las orderlines de productos que no tienen stock disponible.
SELECT orderline.orderinfo_id, orderline.item_id, orderline.quantity
FROM orderline
LEFT JOIN stock ON orderline.item_id = stock.item_id
WHERE stock.item_id IS NULL;
```

- Más información: http://www.postgresqltutorial.com/postgresql-joins/.

Operaciones JOIN

- Con lo visto hasta ahora, que operación JOIN emplear. Esto irá en función de dos aspectos principales.
- **Datos que desea obtener**:
  - Todas las coincidencias → **INNER JOIN**.
  - Todos los registros de una tabla, aunque no haya coincidencias → **LEFT JOIN o RIGHT JOIN**.
  - Todos los registros de ambas tablas, haya o no haya coincidencias → **FULL JOIN**
- **Rendimiento**:
  - **INNER JOIN**: JOIN más eficiente.
  - El resto pueden ser eficientes en ciertos casos, pero tienden a implicar rendimientos más altos.

Operaciones JOIN

- También existe el `NATURAL JOIN`, en todas sus variantes: `NATURAL [INNER, LEFT, RIGHT] JOIN`: http://www.postgresqltutorial.com/postgresql-natural-join/.I
- Implícitamente, busca coincidiencias por nombres de columnas.
- Por tanto, combina dos tablas en función de las columnas que tienen el mismo nombre y tipo de datos, sin especificar la cláusula ON para especificar la condición de combinación.

```
-- Todos los productos en stock junto a su código de barras
SELECT item.item_id, item.description, barcode.barcode_ean
FROM item
NATURAL JOIN stock
NATURAL JOIN barcode

-- Todos los productos independientemente de que haya stock que
--tengan asignado un código de barras.
SELECT item.item_id, item.description, barcode.barcode_ean
FROM item
NATURAL LEFT JOIN stock
NATURAL RIGHT JOIN barcode
```

Operaciones JOIN

- `CROSS JOIN`: Implementa un **producto cartesiano** entre dos o más tablas.
  - Cada fila de A se cruza con todas las filas de B y se almacena la nueva fila resultante. Si hay más columnas se repite la operación.
  - Si tamaño A es M y tamaño B es N el resultado es de dimensión $M \times N$ (!!).
- A diferencia de todos los anteriores, **no incluye cláusula de correspondencia de atributos** (columnas).
- **Mucho cuidado**: Fácilmente genera explosiones de datos en memoria que pueden tirar abajo nuestra consulta, o incluso servidores con recursos limitados.
- Más información: http://www.postgresqltutorial.com/postgresql-cross-join/.

```
SELECT customer.customer_id, customer.fname, orderinfo.orderinfo_id,
orderinfo.shipping
FROM customer
CROSS JOIN orderinfo
ORDER BY customer.customer_id
```

L2.2 Consultas anidadas

Consulta anidada: pertenencia a conjuntos

- Una subconsulta (*subquery*) es una expresión `select-from-where` anidada dentro de otra consulta.
- Suelen aparecer en la cláusula `WHERE` o en la cláusula `FROM`.
- Comprobación de pertenencia: con las cláusulas `IN` y `NOT IN`.

```
-- Mostrar estudiantes con la edad máxima
SELECT *
FROM estudiantes
WHERE edad = (
        SELECT MAX(edad)
        FROM estudiantes);

--Mostrar nombre y apellido de alumnos con alguna nota 'MATRICULA'
SELECT matricula, nombre, apellido
FROM estudiantes
WHERE matricula IN (
        SELECT matricula
        FROM matriculas
        WHERE notatexto = 'MATRICULA');
```

Consulta anidada: relaciones vacías

- Comprobación de relaciones vacías mediante las cláusulas **EXISTS** y **NOT EXISTS**.

```
-- Muestra las órdenes de pedido que tenga el producto 1, siempre
--y cuando haya más de 10 unidades en stock de ese producto.
SELECT *
FROM orderline
WHERE item_id = 1 AND EXISTS(
        SELECT item.item_id, item.description, stock.quantity
        FROM item
        INNER JOIN stock ON stock.item_id = item.item_id
        WHERE stock.quantity > 10 AND stock.item_id = 1);

--Muestra el stock de productos disponibles para ventaja siempre y cuando no haya una orden
--de pedido en proceso que provoque que el stock no esté actualizado. Sucede si no han pasado
--15 días entre la fecha de envio y la fecha de entrega.
SELECT *
FROM stock
WHERE quantity IS NOT NULL AND NOT EXISTS(
        SELECT *
        FROM orderline
        INNER JOIN orderinfo ON orderinfo.orderinfo_id = orderline.orderinfo_id
        WHERE (orderinfo.date_shipped - orderinfo.date_placed) > 12 )
```

Recomendaciones subconsultas WHERE

- Problemas de rendimiento derivados de subconsultas u operaciones involucradas no eficientes (e.g. IN).
- En muchos motores de base de datos, este tipo de consultas obligan a comprobar correspondencias entre cada fila de la tabla origen y todo el posible conjunto de filas de la subconsulta (i.e. *full scan*).
- Casi siempre, una alternativa mucho mejor es realizar la subconsulta en la cláusula `FROM`, eligiendo la operación `JOIN` más adecuada.
- Si usamos una subconsulta con mucha frecuencia puede ser rentable crear una *vista* (veremos más adelante en el curso).

Consulta anidada en FROM

- La idea es usar el resultado de una expresión `select-from-where` dentro de otra, de forma que usemos operaciones sobre conjuntos para cruzarlas.
- Si diseñamos bien la consulta podemos ser mucho más eficientes que usando subconsultas en cláusulas `WHERE`.

```
SELECT Estudiantes.Nombre AS Estudiante, Subconsulta.Asignatura AS AsignaturaConMejorNota
FROM Estudiantes
INNER JOIN (
        SELECT M.Matricula, A.Nombre AS Asignatura
        FROM Matriculas M
        INNER JOIN Asignaturas A ON M.IDAsignatura = A.IDAsignatura
        WHERE M.NotaNumerica = (
                SELECT MAX(NotaNumerica)
                FROM Matriculas
                WHERE Matricula = M.Matricula
        )
) AS Subconsulta ON Estudiantes.Matricula = Subconsulta.Matricula;
```

- Algunas implementaciones requieren renombrar la relación surgida como resultado de la subconsulta.

Consulta anidada en FROM

- No se pueden usar campos de otras tablas de la consulta externa en cláusulas dentro de la subconsulta, con la sintaxis SQL previa a 2003.
- En PostgreSQL se soporta desde la versión 9.3 la cláusula `LATERAL`. Permite exportar los campos del resto de la consulta externa para usarlos dentro de la subconsulta (e.g. en `WHERE`).

```
SELECT idprofesor, nombre, departamento, num_asignaturas
FROM profesores,
        LATERAL( SELECT COUNT(*)
        FROM Asignaturas
        WHERE Asignaturas.idprofesor = profesores.idprofesor) AS
        profe_asignaturas(num_asignaturas)
```

Bibliografía I

- Functions and Operators. Official documentation PostgreSQL v10.

[Silberschatz et al., 2019] Silberschatz, A., Korth, Henry F. Sudarshan, S.
Database System Concepts.
McGraw-Hill, 7th Ed. 2019.