Bases de Datos I Tema 6. SQL: consultas

Martín González Dios

27 de diciembre de 2024

1. Estructura básica de una consulta en SQL

Una consulta en SQL se compone de tres cláusulas principales: **SELECT**, **FROM** y **WHERE**. A continuación, se describen los pasos básicos para construir una consulta:

1. Generar un producto cartesiano de las relaciones especificadas en la **cláusula FROM**.
2. Aplicar los predicados indicados en la **cláusula WHERE** al resultado del paso 1.
3. Para cada tupla del paso 2, extraer los atributos (o resultados de las expresiones) indicados en la **cláusula SELECT**.

- La **cláusula WHERE** de SQL corresponde a la operación **Selección** del álgebra relacional.
- La **cláusula SELECT** de SQL corresponde a la operación **Proyección** del álgebra relacional.

1.1. Ejemplo de consulta

Consideremos la siguiente consulta SQL:

```
SELECT nombre
FROM profesor
WHERE sueldo > 70000 AND nombre_dept = 'Informática';
```

1.2. Unión natural

La unión natural considera únicamente aquellos pares de tuplas que tienen los mismos valores en los atributos que aparecen en los esquemas de ambas relaciones. Por lo tanto, las siguientes consultas son equivalentes:

```
SELECT nombre, asignatura_ID
FROM profesor, enseña
WHERE profesor.ID = enseña.ID;

SELECT nombre, asignatura_ID
FROM profesor NATURAL JOIN enseña;
```

1.3. Uso de JOIN...USING

El constructor de unión natural permite especificar exactamente qué columnas se deben igualar. Por ejemplo:

```
SELECT nombre, nombre_asig
FROM (profesor NATURAL JOIN enseña)
JOIN asignatura USING (asignatura_id);
```

1.4. Eliminación de duplicados

Para controlar la aparición de duplicados en los resultados, se pueden utilizar las siguientes cláusulas:

```
SELECT DISTINCT nombre_dept
FROM profesor;
```

Esta consulta fuerza a que no existan duplicados.

```
SELECT ALL nombre_dept
FROM profesor;
```

Esta consulta muestra duplicados (comportamiento por defecto).

2. Operaciones básicas adicionales en SQL

- **as**: puede aparecer en SELECT y/o en FROM. Se usa si se desea cambiar el nombre de un atributo.
  SELECT nombre as nombre_profesor...
- **like**: comparación de patrones (en texto).
  WHERE edificio like '%Watson%';
  - 'Intro%': coincide con cualquier cadena de caracteres que empiece con Intro.
  - '%Infor%': coincide con cualquier cadena de caracteres que contenga Infor.
  - '___': coincide con cualquier cadena de caracteres que tenga exactamente 3 caracteres.
  - '___%': coincide con cualquier cadena de caracteres que tenga, al menos, 3 caracteres.
- *****: se puede usar en SELECT para indicar 'todos los atributos'.
- **order by**: de manera predeterminada coloca los elementos de forma ascendente.
  SELECT * from profesor order by sueldo desc, nombre asc; (sueldo en forma descendente, si dos profesores tienen el mismo sueldo, se ordenan en orden ascendente de nombre).
- **between**: WHERE sueldo between 90000 and 120000;

3. Operaciones sobre conjuntos en SQL

- **union**: (consulta1) union (consulta2); que estén en la consulta1 o en la consulta2, o en ambas (union all para mantener duplicados).
- **intersect**: (consulta1) intersect (consulta2); que estén en la consulta1 y en la consulta2 (intersect all para mantener duplicados).
- **except**: todas las tuplas de la primera entrada que no están en la segunda (except all para mantener duplicados).

Las **tablas** deben ser **compatibles** (sintácticamente): mismo número de atributos y mismos tipos de dato de cada atributo.

4. Valores nulos

Se trata con **unknown** el resultado de cualquier comparación en la que intervenga el valor null, tratándose como un tercer valor lógico (true, false, unknown).

- **and**:
  - true and unknown: unknown
  - false and unknown: false
  - unknown and unknown: unknown
- **or**:
  - true or unknown: true
  - false or unknown: unknown
  - unknown or unknown: unknown
- **not**: not unknown: unknown

4.1. Ejemplo de uso de null

```
SELECT nombre
FROM profesor
WHERE sueldo is (not) null;
```

Que el sueldo sea (o no) nulo. También existe is unknown e is not unknown.

5. Funciones de agregación en SQL

- Media(**avg**)
- Mínimo (**min**)
- Máximo (**max**)
- Total (**sum**)
- Recuento (**count**)

Para elminiar duplicados antes de usar una función de agregación se usa **distinct**: select count distinct ID from...

Para contar el número de tuplas de una relación: select count * from asignatura;

5.1. Group by

Se usa para formar grupos. Las tuplas con el mismo valor en todos los atributos en la cláusula group by se sitúan en un grupo.

```
SELECT nombre_dept, avg(sueldo) as med_sueldo
FROM profesor group by nombre_dept;
```

En este caso el group by agrupa los resultados por el nombre del departamento (nombre_dept). Esto significa que el promedio de sueldo se calculará para cada departamento por separado.

Cualquier atributo que no esté en la cláusula group by **solo debe aparecer** dentro de una función de agregación si aparece también en la cláusula SELECT. Es decir, **solo los atributos que aparecen en la sentencia SELECT sin agregarse se encuentran en la cláusula group by**.

5.2. Having

Indicar una condición que se aplica a grupos y no a tuplas.

```
SELECT nombre_dept, avg(sueldo) as med_sueldo
FROM profesor group by nombre_dept having avg(sueldo) > 50000;
```

5.3. Secuencia de operaciones para crear consultas con cláusulas de agregación

1. La **cláusula FROM** se evalúa en primer lugar.
2. Si existe una **cláusula WHERE**, el predicado de esta cláusula se aplica al resultado de la relación de la cláusula FROM.
3. Las tuplas que satisfagan el predicado WHERE se sitúan en grupos de acuerdo a la **cláusula group by**, si existe. Si no existe todo **el conjunto de tuplas se trata como si fuese un único grupo**.
4. Se aplica la **cláusula having**, si existe, a cada uno de los grupos; los grupos que no satisfagan el resultado de la cláusula having se eliminan.
5. La **cláusula SELECT** utiliza los grupos que quedan para generar tuplas con el resultado de la consulta aplicando las funciones de agregación **para obtener una única tupla resultado para cada grupo**.

5.4. Agregación con valores nulos y booleanos

Todas las funciones de agregación excepto count(*) deben **ignorar los valores nulos** que aparezcan como entrada.
Count de una **colección vacía** debe valer 0, el resto de operaciones de agregación devuelven null cuando se aplican a una colección vacía.

6. Subconsultas anidadas

6.1. in

Comprueba la **pertenencia a un conjunto**, donde el conjunto es la colección de valores resultados de una cláusula SELECT.

```
SELECT distinct asignatura_id
FROM seccion
WHERE semestre = 'Otoño' and
año = 2009 and
    asignatura_id in (SELECT distinct asignatura_id
    WHERE semestre = 'Primavera' and año = 2010);

...WHERE nombre not in ('Juanlu', 'Mozart');
```

6.2. > some

Mayor que al menos una.
Ejemplo: Encontar los nombres de todos los profesores cuyo sueldo es mayor que al menos uno de los profesores del departamento de Biología.

```
SELECT nombre
FROM profesor
WHERE sueldo >some (SELECT sueldo
                    FROM profesor
                    WHERE nombre_dept = 'Biología');
```

También existen ¡some, <=some, <>some, ...
=some es igual que in, pero <>some NO es igual que not in.

6.3. >all

Superior al de todos.
Ejemplo: todos los profesores que tienen un sueldo mayor al de cualquier profesor del departamento de Biología.

```
SELECT nombre
FROM profesor
WHERE sueldo >all (SELECT sueldo
                    FROM profesor
                    WHERE nombre_dept = 'Biología');
```

También existen <all, <=all, <>all, ...
<>all es igual que not in, pero =all NO es igual que in.

6.4. exists

Devuelve el valor **true si su argumento de subconsulta no resulta vacía**.
Ejemplo: encontrar todas las asignaturas que se enseñaron tanto en el semestre de de otoño de 2009 como en el semestre de primavera de 2010.

```
SELECT asignatura_id
FROM sección as S
WHERE semestre = 'Otoño' and año = 2009 and
    exists (SELECT *
            FROM sección as T
            WHERE semestre = 'Primavera' and año = 2010
            and S.asignatura_id = T.asignatura_id);
```

Para escribir que la relación A contiene a la relación B se ecribe de la siguiente forma:

```
not exists(B except A)
```

6.5. unique

Devuelve un valor true si la subconsulta que se le pasa como argumento **no contiene tuplas duplicadas**.
Ejemplo: encontrar todas las asignatura que se ofertaron al menos una vez en 2009.

```
SELECT T.asignatura_id
FROM asignatura as T
WHERE unique (SELECT R.asigntura_id
              FROM sección as R
              WHERE T.asignatura_id = R.asignatura_id and
              R.año = 2009);
```

6.6. with

Definir **vistas temporales** cuya definición solo está disponible para la consulta en la que aparece la cláusula.
Ejemplo: seleccionar los departamentos con el máximo presupuesto.

```
with max_presupuesto(valor) as
    (SELECT max(presupuesto)
    FROM departamento)

SELECT presupuesto
FROM departamento, max_presupuesto
WHERE departamento.presupuesto = max_presupuesto.valor;
```

6.7. Subconsultas escalares

Se permiten escribir en cualquier punto en el que **una expresión devuelve un valor**.
Ejemplo: listar todos los departamentos junto con el número de profesores de cada departamento.

```
SELECT nombre_dept
    (SELECT count(*)
    FROM profesor
    WHERE departamento.nombre_dept =
        profesor.nombre_dept)
    as num_profesores
FROM departamento;
```

7. Expresiones de reunión en SQL

7.1. on

Permite la reunión de un predicado general sobre relaciones. Ejemplo:

```
SELECT *
FROM estudiante join matricula on
    estudiante.ID = matricula.ID;
```

Esta consulta sería igual que un natural join, excepto que **contiene los atributos de ID 2 veces en la reunión**.
Para que solo se mostrara una vez se podría filtrar con el SELECT.
on se puede utilizar por 2 motivos: para hacer reuniones externas y por legibilidad.

7.2. Reunión externa

Funciona de forma similar a la operación de reunión, **creando tuplas en el resultado que contienen valores nulos**.
Existen 3 formas:

- **Reunión externa izquierda** (left outer join): conserva las tuplas solo en la relación de antes (a la izquierda) de esta operación.
- **Reunión externa derecha** (right outer join): conserva las tuplas solo en la relación de después (a la derecha) de esta operación.
- **Reunión externa completa** (full outer join): conserva las tuplas de ambas relaciones.

Ejemplo:

```
SELECT *
FROM estudiante natural left outer join matrícula;
```

```
SELECT *
FROM estudiante left outer join matrícula
    on estudiante.ID = matrícula.ID;
```

Estas 2 consultas son iguales, excepto que la segunda el atributo ID estaría duplicado.

On y where se comportan de manera **diferente** en la **reunión externa**. La condición on forma parte de la especificación de los **join (reuniones), que por defecto son inner, internas**.