Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ingeniería en Computación

Bases de Datos

Tarea 14

Silverio Martínez Andrés

21/04/2024

TAREA 14

Investigación (Casos de uso (con ejemplos) y restricciones de sentencias del lenguaje de consultas de datos)

1. SELECT

Esta sentencia, en SQL funciona principalmente para la recuperación de información en una base de datos. Se caracteriza por tener la siguiente estructura:

```
SELECT [columna1, columna2, ...]
FROM [nombreTabla] <condicion>
```

- **Casos de uso**

  Entre los casos de uso más comunes de la sentencia "SELECT", podemos encontrar los siguientes:

  - **Recuperación de datos**:
    Siendo de los casos más comunes, utilizado para recuperar los datos específicos de una o varias tablas.

    ```
    SELECT * FROM empleados;
    ```

    Donde, en la tabla "empleados", se mostrarán todos los datos de las "nxm" filas/columnas que tenga la tabla
  - **Recuperación de datos por columna**:
    Cambiando el asterisco puesto anteriormente, y poniendo específicamente una columna de la tabla, podemos solamente obtener los valores de dicha columna

    ```
    SELECT idEmpleado FROM empleados;
    ```

    Donde solamente devolverá la columna "idEmpleado" de la tabla "empleados"
  - **Operaciones con los datos**:
    Se pueden llegar a realizar diversas operaciones (suma, contar, etc)

    ```
    SELECT COUNT(*) FROM ventas;
    ```

    Donde por medio de esta consulta, se devolverán el número de filas de la tabla "ventas"

- **Restricciones**

  Entre las restricciones que tiene la sentencia "SELECT", encontramos:

- **Integridad de datos**:
  La consulta realizada debe respetar la integridad de los datos para evitar resultados inesperados o errores
- **Permisos de acceso**:
  El usuario que realiza la consulta debe tener permisos suficientes para acceder a las tablas y columnas deseadas

2. FROM

Esta sentencia, en SQL, se utiliza principalmente para especificar la tabla de la cual se van a recuperar datos de una consulta SELECT. Su sintaxis es la siguiente:

```
... FROM [nombreTabla]
```

- ## Casos de uso

  El caso de uso principal de la sentencia "FROM" es:

  - **Referenciación de tabla**:
    Donde se especifica la tabla de la cual se desean recuperar los datos

    ```
    SELECT * FROM empleados;
    ```

    Esto recuperará todos los datos de la tabla "empleados"

- ## Restricciones

  Entre las restricciones que tiene la sentencia "FROM", encontramos:

  - **Tabla existente**:
    La tabla que se especifique en el FROM, debe de existir en nuestra base de datos
  - **Sintaxis correcta**:
    El FROM siempre irá entre las columnas del SELECT y la tabla en la que se van a buscar los datos de las columnas

3. **JOIN**

La sentencia JOIN, en SQL, sirve para poder combinar filas de dos o más tablas basadas en una condición relacionada entre ellas. Su sintaxis es la siguiente:

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

- ## Casos de uso

  La sentencia JOIN, como ya se mencionó anteriormnete, sirve para combinar filas de dos o más tablas de acuerdo a una condición que las relaciona, pero también hay diferentes JOIN´s que pueden ser usados, siendo los más relevantes:

- **INNER JOIN**:
  Combina las filas de dos tablas donde la condición especificada se cumple en ambas tablas, como se muestra en el siguiente ejemplo:

```
SELECT orden.idOrden, cliente.nomCliente
FROM orden
INNER JOIN clientes
ON orden.idClientes = clientes.idClientes;
```

  Donde, por medio del INNER JOIN, se combinan las tablas "orden" y "clientes", basandose en el "idCliente"
- **LEFT JOIN**:
  Este tipo de JOIN va a devolver las filas de la tabla izquierda y las filas que coinciden de la tabla derecha. Si no hay coincidencias, devolverá un NULL. Ejemplo de esto es:

```
SELECT empleados.nombre, departamentos.nombre
FROM empleados
LEFT JOIN departamentos
ON empleados.idDepartamento = departamentos.id;
```

  En este ejemplo, se devolveran todas las filas de la tabla "empleados", y si un empleado no tiene un departamento asignado, la columna de nombre del departamento regresará NULL
- **RIGHT JOIN**:
  Este tipo de JOIN va a devolver las filas de la tabla derecha y las filas que coinciden de la tabla izquierda. Si no hay coincidencias, devolverá un NULL. Con el mismo ejemplo del inciso anterior:

```
SELECT empleados.nombre, departamentos.nombre
FROM empleados
LEFT JOIN departamentos
ON empleados.idDepartamento = departamentos.id;
```

  Ahora, en este caso, se devolveran todas las filas de la tabla "departamentos", y si un departamento no tiene emplados asignados, la columna de nombre del empleado regresará NULL
- **FULL JOIN**:
  Este tipo de JOIN va a devolver todos los valores de las filas cuando existe una coincidencia en alguna de las tablas en las que está comparando. Teniendo el mismo ejemplo, pero con FULL JOIN:

```
SELECT empleados.nombre, departamentos.nombre
FROM empleados
FULL JOIN departamentos
ON empleados.idDepartamento = departamentos.id;
```

  En este caso, el ejemplo devolverá todas las filas, con sus respectivos datos de ambas tablas cuando haya una coincidencia en en

"idDepartamento" y mostrando NULL donde no haya coincidencias

- **Restricciones**

  Entre las restricciones que existen para los diversos tipos de JOIN, encontramos:

  - **Uniones erroneas**
    Las columnas especificadas en la condición de unión, si o si, deben de ser el mismo tipo de datos entre sí, es decir, un varchar(12), solo podrá hacer JOIN con otro varchar(12)
  - **Rendimiento**
    Si se llegasen a unir grandes conjuntos de datos, esto podría provocar afectaciones en el rendimiento de la consulta

4. WHERE

La sentencia WHERE, en SQL, se utiliza para filtrar filas basadas en una condición específica en una consulta SELECT. Su sintaxis es la siguiente:

```
... WHERE <condicion>
```

- **CASOS DE USO**

  Como ya se mencionó, la sentencia WHERE se utiliza para filtrar datos, entre las opciones para la filtración de datos, podemos encontrar:

  - **Filtrar por valores específicos**:
    Se especifica una condición para poder filtrar las filas que cumplen con esa misma condición, por ejemplo:

    ```
    SELECT *
    FROM empleados
    WHERE departamento = 'Ventas';
    ```

    Donde, en este caso, se devolverán todas las filas de la tabla "empleados", en donde el valor de la columna "departamento" sea igual a "Ventas"
  - **Filtrar por rangos**:
    Para ello, podemos utilizar diferentes operadores aritméticos como "¿", "¡" o BETWEEN para filtrar por un rango de valores, por ejemplo:

    ```
    SELECT *
    FROM productos
    WHERE precio > 100 AND precio < 200;
    ```

    En este ejemplo, se mostraran todas las filas de la tabla "productos" donde el precio está entre 100 y 200

- **Filtrar por patrones de texto**:
  Mediante el uso de un operador LIKE, podemos buscar patrones de texto dentro de una columna, por ejemplo:

```
SELECT *
FROM clientes
WHERE nombre LIKE 'M%';
```

  Para este ejemplo, se devolverán todas las filas de la tabla "cliente" donde el nombre comienza con la letra "M"

- ## Restricciones

  Entre las restricciones que existen para la sentencia WHERE, encontramos:

  - **Consulta con gran número de resultados**:
    Por ejemplo, al hacer un mal uso de LIKE "porcentaje", podría regresar una gran cantidad de datos y hacer que el rendimiento baje

5. HAVING

La sentencia HAVING es similar a utilizar un WHERE, pero es utilizada específicamente para filtrar resultados después de que se han agrupado mendiante un GROUP BY, siendo fundamental cuando se desea aplicar condiciones a grupos de filas. Su sintaxis es la siguiente:

```
SELECT column1, function(column2)
FROM table
GROUP BY column1
HAVING condition;
```

- ## Casos de uso

  Entre los casos de uso más comunes para la sentencia GROUP BY, podemos encontrar:.

  - **Filtrar grupos por una función de agregación**:
    Por ejemplo, deseamos saber que departamentos tienen un total de ventas superior a un cierto criterio, del ejemplo:

```
SELECT departamento, SUM(ventas) AS total_ventas
FROM registros_ventas
GROUP BY departamento
HAVING SUM(ventas) > 10000;
```

    Donde, en el ejemplo, se agrupan las ventas por departamneto y solo muestra aquellos departamentos donde el total de ventas es mayor a 10000

  - **Aplicar condiciones a múltiples funciones de agregación**
    Ahora, por ejemplo, si se quiere identificar los departamentos donde el número medio de ventas por empleado excede un valor específico, se puede hacer lo siguiente:

```
SELECT departamento, AVG(ventas) AS promedio_ventas
FROM registros_ventas
GROUP BY departamento
HAVING AVG(ventas) > 1500;
```

Donde, en este ejemplo, se mostrarán los departamentos donde el promedio de ventas por empleado es mayor a 1500

- **Restricciones**

  La sentencia HAVING, puede presentar las siguientes restricciones:

  - **Utilización con GROUP BY**
    La sentencia HAVING funciona de mejor forma con un GROUP BY, en cambio, si no se utiliza con uno, el HAVING funcionaría como un WHERE cualquiera
  - **Limitaciones**
    Las condiciones de HAVING deben de involucrar funciones de agregación si se utilizan para filtrar grupos y no filas individuales
  - **Orden de procesamiento**
    La sentencia HAVING se procesa después de la sentencia GROUP BY, lo que significa que no pueden utilizar alias de columnas definidos en la cláusula SELECT si estos alias no existían antes de GROUP BY

6. CORRELACIONADAS

Las subconsultas correlacionadas en SQL son subconsultas en las que la subconsulta depende de la consulta externa. Es decir, la subconsulta se ejecuta una vez para cada fila devuelta por la consulta externa. Su sintaxis es similar a cualquier otra subconsulta, pero hace referencia a las columnas de la consulta externa dentro de la subconsulta, ejemplo de esto es el siguiente:

```
SELECT columna1, columna2, ...
FROM tabla1
WHERE columna
IN (SELECT columna FROM tabla2 WHERE condicion);
```

- **Casos de uso**

  Algunos casos de uso que puede presentar esta subconsulta podrían ser:

  - **Filtrar resultados basados en una consulta externa**:
    Se utiliza una subconsulta que hace referencia a columnas de la consulta externa para filtrar resultados.

    ```
    SELECT nombre
    FROM empleados
    WHERE salario
    > (SELECT AVG(salario) FROM empleados);
    ```

Esto recuperará los nombres de los empleados cuyo salario sea mayor que el salario promedio de todos los empleados.

- **Restricciones**

  Entre sus restricciones más comunes podemos encontrar:

  - **Complejidad de lectura**
    Debido a la dependencia contextual y la estructura anidada, puede provocar que las consultas realizadas sean más dificiles de entender
  - **Uso de índices**
    Es importante que las tablas involucradas en las subconsultas correlacionadas estén correctamente indexadas. De lo contrario, el rendimiento puede degradarse significativamente