Bases de Datos I
Unidad VIII

Lenguaje de Definición de Datos
SQL/DML

SELECT

https://www.flaticon.com/free-icons/relations

Funciones de agregación

- **A veces no necesitamos los datos de una o varias tablas, sino resúmenes o caracterizaciones sobre los mismos.**
- **¿Cuántas facturas se realizaron en el mes de Marzo de 2023?**
- **¿Cuál es el gasto promedio que realizan los clientes con cada compra?**

- **La sentencia SELECT permite consultar este tipo de datos, haciendo uso de las funciones de agregación.**

Funciones de agregación

- **SQL provee distintas funciones de agregación:**
  - **COUNT** → Calcula la cantidad de filas
  - **MAX** → Obtiene el valor máximo
  - **MIN** → Obtiene el valor mínimo
  - **SUM** → Calcula la sumatoria
  - **AVG** → Calcula la media aritmética
  - **STD** → Calcula la desviación estándar
  - **EVERY** → Retorna verdadero si todas cumplen

Ejemplo

- **¿Cómo obtenemos la cantidad de facturas que se realizaron en el mes de Marzo de 2015?:**

```
SELECT COUNT(*) FROM factura
WHERE fecha BETWEEN '2015-03-01'
AND '2015-03-31';
```

- **El resultado no son las filas de la tabla factura, es sólo la cantidad (una fila de una sola columna) de filas que cumplen la condición.**

Ejemplo

- **¿Cuál es el precio promedio de los productos?**

  SELECT AVG(precio) **FROM** producto;

- **¿Cuál es el precio máximo que tiene algún producto?**

  SELECT MAX(precio) **FROM** producto;

- **¿Cuál es el precio mínimo que tiene algún producto?**

  SELECT MIN(precio) **FROM** producto;

Ejemplo

- **¿Cuál es el importe total facturado durante el año 2015?**

```
SELECT SUM(l.cantidad * p.precio)
FROM factura AS f, linea AS l, producto AS p
WHERE f.numero = l.factura
   AND l.producto = p.codigo
   AND EXTRACT(YEAR FROM fecha) = 2015;
```

Ejemplo

- **También es posible utilizar una variante de la función COUNT que permite realizar la cuenta de filas que presenten diferencias en algunas de sus columnas:**

```
SELECT COUNT(DISTINCT producto)
FROM linea AS l, factura AS f
WHERE l.factura = f.numero
    AND fecha = '2015-05-05';
```

Agrupamiento

- **Si bien las funciones de agregación son de gran utilidad, a veces no son suficiente.**
- **¿Cómo hacemos para obtener por cada año la facturación total del mismo?**
- **¿Cómo hacemos para obtener cada número de factura con la cantidad de lineas que tiene cada una?**
- **Para esto necesitamos realizar agrupamientos de filas.**

Agrupamiento

- **La sentencia SELECT de SQL presenta la cláusula GROUP BY, que permite realizar agrupamiento de filas.**

```
SELECT *
FROM tabla
GROUP BY [columnas/expresiones];
```

- **Combinando el agrupamiento y las funciones de agregación, podemos responder las preguntas.**

Ejemplo

- **¿Cómo hacemos para obtener cada año, y la facturación del mismo?**

```
SELECT EXTRACT(YEAR FROM fecha),
                    SUM(l.cantidad * p.precio)
FROM factura AS f, linea AS l, producto AS p
WHERE f.numero = l.factura
    AND l.producto = p.codigo
GROUP BY EXTRACT(YEAR FROM fecha);
```

Ejemplo

- **¿Cómo hacemos para obtener cada número de factura con la cantidad de lineas que tiene cada una?**

```
SELECT f.numero, COUNT(*)
FROM factura AS f, linea AS l
WHERE f.numero = l.factura
GROUP BY f.numero;
```

Ejemplo

- **El agrupamiento, sin agregación:**

```
SELECT l.factura
FROM linea AS l
GROUP BY l.factura;
```

- **Eliminación de repetidos:**

```
SELECT DISTINCT l.factura
FROM linea AS l;
```

• Selección sobre grupos

- **En ocasiones utilizamos funciones de agregación realizando agrupamiento, pero sólo queremos obtener ciertos grupos que cumplan una determinada condición.**

- **Es claro que queremos realizar una selección, pero la cláusula WHERE no nos sirve, ya que determina que filas van a ser incluidas y luego se realiza el agrupamiento de las mismas.**

Selección sobre grupos

- **La sentencia SELECT nos permite realizar selección sobre el agrupamiento a través de la clausula HAVING:**

```
SELECT *
FROM tabla
GROUP BY [columnas/expresiones]
HAVING condición;
```

Ejemplo

- **Obtenemos el número de las facturas que tienen más de 3 lineas:**

```
SELECT f.numero
FROM factura AS f, linea AS l
WHERE f.numero = l.factura
GROUP BY f.numero
HAVING COUNT(*) > 3;
```

Subconsultas

- **En la sentencia SELECT es posible realizar el anidamiento de consultas, consultas dentro de consultas.**

- **Es posible utilizar subconsultas en la proyección (SELECT) o en las selecciones (WHERE, HAVING).**

Ejemplo

- **Queremos obtener el código y la descripción de los productos que tienen el precio más alto.**

```
SELECT codigo, descripcion
FROM producto
WHERE precio = (SELECT MAX(precio)
                FROM producto)
```

- **En este caso utilizamos una subconsulta en la selección (WHERE)**

Subconsultas correlacionadas

- **Dentro de las subconsultas, es posible hacer referencia a la fila actual de la consulta que la contiene.**
- **Este tipo de referencias se denomina, referencia externa.**
- **Este tipo de subconsulta se denomina, correlacionada.**
- **La subconsulta se ejecuta una vez por cada fila de la consulta contenedora.**

Ejemplo

- **Queremos obtener el número de factura y el monto total de la misma.**

```
SELECT numero, (SELECT
                    SUM(l.cantidad * p.precio)
                FROM linea AS l, producto AS p
                WHERE l.factura = f.numero
                AND l.producto = p.codigo)
FROM factura AS f;
```

Subconsultas

- **El lenguaje incluye una serie de operadores ideales para utilizar con subconsultas:**
  - **IN →** Permite comprobar pertenencia
  - **EXISTS →** Permite verificar existencia
  - **SOME →** Permite verificar relación con algunos
  - **ANY →** Sinónimo de SOME
  - **ALL →** Permite verificar relación con todos

IN / NOT IN

- **Utilización:**
  - **x IN (...): El valor debe estar presente en los valores para que sea verdadero.**
  - **x NOT IN (...): El valor no debe estar presente en los valores para que sea verdadero.**
- **No solo puede ser utilizado con subconsultas, también con listerales:**

```
SELECT * FROM factura
WHERE tipo IN ('A', 'B');
```

Ejemplo

- **Vamos a obtener el apellido y nombre de clientes que tengan por lo menos una factura:**

```
SELECT apellido, nombre
FROM cliente AS c
WHERE c.documento IN (SELECT f.cliente
                      FROM factura);
```

EXISTS / NOT EXISTS

- **Utilización:**
  - **EXISTS (...): La lista de valores debe contener algún elemento para que sea verdadero.**
  - **NOT EXISTS (...): La lista de valores debe estar vacía para que sea verdadero.**

Ejemplo

- **Vamos a obtener el apellido y nombre de clientes que tengan por lo menos una factura:**

```
SELECT apellido, nombre
FROM cliente AS c
WHERE EXISTS (SELECT f.cliente
                FROM factura
                  WHERE f.cliente = c.documento);
```

SOME / ANY

- **Utilización:**
  - **x = ANY (...): El valor debe coincidir con alguno de los valores para que sea verdadero.**
  - **x <> ANY (...): El valor no debe coincidir con alguno de los valores para que sea verdadero.**
  - **x > ANY (...): El valor debe ser mayor que alguno de los valores para que sea verdadero.**
  - **x < ANY (...): El valor debe se menor que alguno de los valores para que sea verdadero.**
  - **x >= ANY (...): El valor debe se mayor o igual que alguno de los valores para que sea verdadero.**
  - **x <= ANY (...): El valor debe se menor o igual que alguno de los valores para que sea verdadero.**

Ejemplo

- **Vamos a obtener el apellido y nombre de los clientes que hayan nacido, cuando se haya realizado alguna factura:**

```
SELECT apellido, nombre
FROM cliente
WHERE nacimiento = SOME (SELECT fecha
                         FROM factura);
```

ALL

- **Utilización:**
  - **x = ALL (...): El valor debe coincidir con todos los valores para que sea verdadero.**
  - **x <> ALL (...): El valor no debe coincidir con ninguno de los valores para que sea verdadero.**
  - **x > ALL (...): El valor debe ser mayor que todos los valores para que sea verdadero.**
  - **x < ALL (...): El valor debe se menor que todos los valores para que sea verdadero.**
  - **x >= ALL (...): El valor debe se mayor o igual que todos los valores para que sea verdadero.**
  - **x <= ALL (...): El valor debe se menor o igual que todos los valores para que sea verdadero.**

Ejemplo

- **Vamos a obtener el apellido y nombre de los clientes que hayan nacido, antes que se realizaran todas las facturas:**

```
SELECT apellido, nombre
FROM cliente
WHERE nacimiento < ALL (SELECT fecha
                        FROM factura);
```

Equivalencias

- **Algunas construcciones utilizando los operadores, son equivalentes entre si:**

Expresión 1	**Expresión2**
X = **SOME** (...)	X **IN** (...)
X = **ANY** (...)	X **IN** (...)
X <> **ALL** (...)	X **NOT IN** (...)
EXISTS (subconsulta relacionada X)	X **IN** (...)
NOT EXISTS (subconsulta relacionada X)	X **NOT IN** (...)

Y SQL también tiene Vistas

- **Las vistas pueden ser consideradas como tablas virtuales o consultas almacenadas.**
- **Su contenido es el resultado de una sentencia SELECT.**
- **Su contenido no se encuentra almacenado, sino que se evalúa cada vez que son utilizadas.**
- **Pueden ser utilizadas como cualquier tabla normal.**

CREATE VIEW

- **Como cualquier otro elemento de la base de datos, se crean utilizando la sentencia CREATE.**

```
CREATE VIEW activo(d, a, n) AS
SELECT documento, apellido, nombre
FROM empleado
WHERE retiro IS NOT NULL;
```

Consultas sobre vistas

- **Una vez creada la vista, podemos realizar consultas sobre ella, como si se tratase de cualquier tabla.**

```
SELECT * FROM activo;
```

- **Incluso participar de reuniones.**

```
SELECT *
FROM activo, cargo
WHERE activo.cargo = cargo.codigo;
```

ALTER VIEW

- **Como cualquier otro elemento de la base de datos, se modifican con una sentencia ALTER.**

```
ALTER VIEW activo
RENAME TO empleado_activo;
```

DROP VIEW

- **Como cualquier elemento de la base de datos, se eliminan con una sentencia DROP.**

  DROP VIEW empleado_activo;

- **Si tenemos elementos que referencian la vista, podemos utilizar la eliminación en cascada.**

  DROP VIEW empleado_activo **CASCADE;**

Vistas modificables

- **Aunque las vistas, sean el resultado de una consulta, pueden ser modificables.**
- **Si se modifica la vista, se modifica entonces la tabla de la que surge la vista.**
- **Si no se respetan ciertas restricciones, las vistas son de solo consulta (aunque se pueden hacer modificables también utilizando disparadores).**

Vistas modificables

- **El estandar nos indica que se deben cumplir las siguientes restricciones para que una vista sea modificable:**
  - **La sentencia SELECT debe ser sobre una única tabla (FROM), y no puede contener DISTINCT.**
  - **Si un atributo de la tabla no esta en la vista, debe soportar valores NULL o valores por defecto (DEFAULT).**
  - **Si la vista está sobre la tabla T, las subconsultas no pueden referirse a T, pero si a otras tablas.**
  - **En una consulta, no puede utilizarse GROUP BY o funciones de agregación (AVG, SUM, COUNT, etc).**

Vistas modificables

- **La opción WITH CHECK OPTION permite verificar que las filas insertadas o modifcadas, sean visibles en la vista.**

```
CREATE VIEW activo(d, a, n) AS
SELECT documento, apellido, nombre
FROM empleado
WHERE retiro NOT IS NULL
WITH CHECK OPTION;
```

¿Para qué sirven las vistas?

- **Realizar consultas complejas más fácilmente.**
- **Para proporcionar compatibilidad con versiones anteriores.**
- **Como mecanismo de seguridad.**

Bibliografía

- **SQL-99 Complete, Really. 1999. Peter Gulutzan y Trudy Pelzer.**
- **PostgreSQL Introduction and Concepts. 2001. Momjian, Bruce.**
- **PostgreSQL 9.6 Documentation. 2016. The PostgreSQL Global Development Group.**
- **A Simpler (and Better) SQL Approach to Relational Division. 1998. Victor M. Matos y Rebecca Grasser.**