



Universidad Nacional de La Matanza
Catedra de Base de Datos

Clase Teórica de SQL – Parte 2

Lunes 26/10/2020



Funciones de agregación:

Se aplican sobre un conjunto de filas y devuelven un único valor para todas ellas.

Hay muchas, las que mas se utilizan son:

- SUM()
- MAX()
- MIN()
- AVG()
- COUNT()



SUM()

Calcula la suma de valores de una columna.

Ejemplo:

EMPLEADO (legajo, nom, ape, salario, categoría, tel, cod_depto)
DEPARTAMENTO (cod_depto, descripcion)

Suma de salarios de todos los empleados

```
SELECT SUM(salario)
FROM Empleado
```



SUM()

Calcula la suma de valores de una columna.

Ejemplo2:

EMPLEADO (legajo, nom, ape, salario, categoría, tel, cod_depto)
DEPARTAMENTO (cod_depto, descripcion)

Suma de salarios de todos los empleados de Sistemas

```
SELECT SUM(salario)
FROM Empleado e, Departamento d
WHERE e.cod_depto = d.cod_depto
AND d.descripcion = 'Sistemas'
```



MAX() Devuelve el mayor valor de una columna

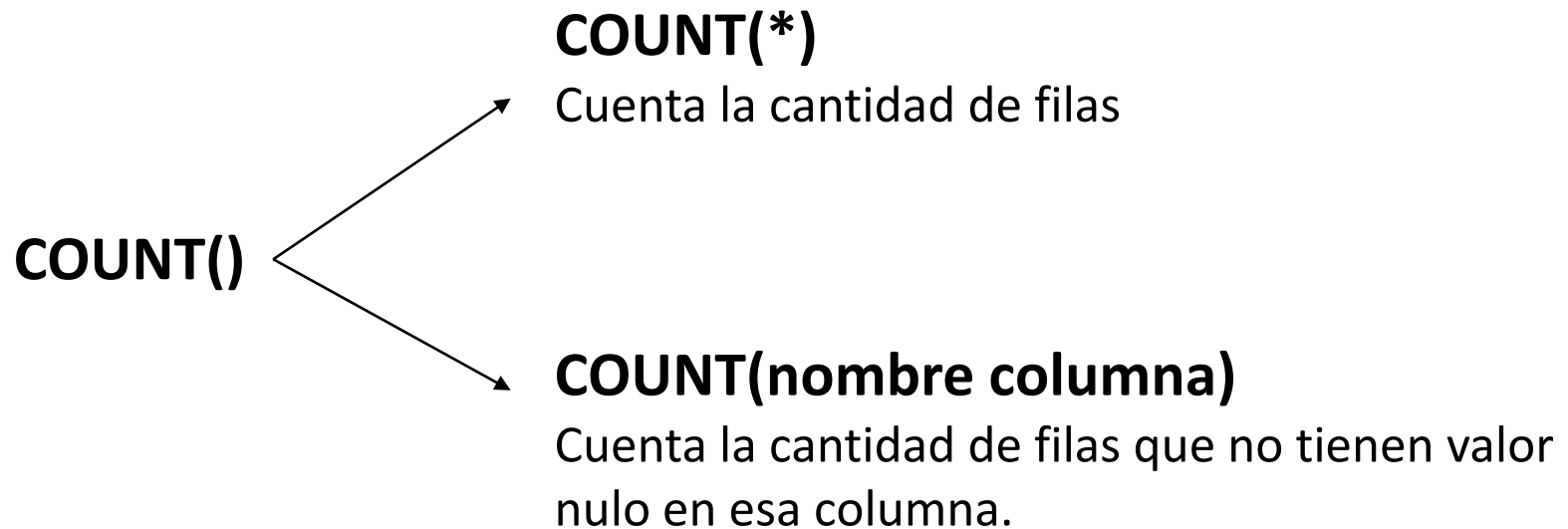
MIN () Devuelve el menor valor de una columna

Ejemplo:

EMPLEADO (legajo, nom, ape, salario, categoría, tel, cod_depto)
DEPARTAMENTO (cod_depto, descripcion)

Salario Mínimo y Máximo de los empleados de categoría 3

```
SELECT MIN(salario), MAX(salario)
FROM Empleado
WHERE categoria = 3
```





COUNT()

Ejemplo:

EMPLEADO (legajo, nom, ape, salario, categoría, tel, cod_depto)
DEPARTAMENTO (cod_depto, descripcion)

Cantidad de Departamentos

```
SELECT COUNT(*)  
FROM Departamento
```



COUNT()

Ejemplo2:

EMPLEADO (legajo, nom, ape, salario, categoría, tel, cod_depto)
DEPARTAMENTO (cod_depto, descripcion)

Cantidad total de Empleados y cuantos de ellos tienen teléfono

```
SELECT COUNT(*) cant_empleados,  
        COUNT(tel) cant_con_tel  
FROM Empleado
```




GROUP BY

Permite dividir el resultado de una consulta en grupos, según el valor de uno o mas atributos.

Ejemplo:

EMPLEADO (legajo, nom, ape, salario, categoría, tel, cod_depto)
DEPARTAMENTO (cod_depto, descripcion)

Contar la cantidad de empleados de cada categoría

```
SELECT categoria, COUNT(*)  
FROM Empleado  
GROUP BY categoria
```



GROUP BY

```
SELECT categoria, COUNT(*)  
FROM Empleado  
GROUP BY categoria
```

Las columnas normales (no agrupadas)
siempre **DEBEN** estar en el GROUP BY



GROUP BY

```
SELECT COUNT(*)  
FROM Empleado  
GROUP BY categoria
```

Pero en el GROUP BY es posible colocar
columnas que no estén en el SELECT



HAVING

Permite especificar condiciones que se aplicarán luego del GROUP BY.

Es como el WHERE pero para colocar condiciones luego de haber agrupado.

Las condiciones del WHERE se aplican antes de hacer el agrupamiento.



HAVING

Ejemplo:

Listar los Departamentos que tengan 5 o más empleados de categoría 2.

```
SELECT cod_depto, count(*) cant
FROM Empleado
WHERE categoria = 2
GROUP BY cod_depto
HAVING count(*) >=5
```

Esta condición se
resuelve **antes** del
GROUP BY

Esta condición se
resuelve **después** del
GROUP BY

NO se puede usar el Alias de la columna
HAVING cant >= 5



ORDER BY

Permite ordenar el resultado de una consulta por una o mas columnas.

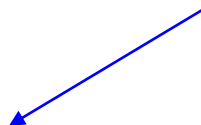
Se debe colocar al **final** de la consulta.

SELECT

FROM

.....

Por defecto ordena en
forma Ascendente



ORDER BY columna1, columna2 **DESC**,, columnaN **ASC**



ORDER BY

- Se puede ordenar por una columna que no se encuentre en el SELECT (siempre y cuando la consulta no esté agrupada)

R (a, b, c)

```
SELECT a, b  
FROM R  
ORDER BY c
```



ORDER BY

- Se puede usar el alias de una columna en el ORDER BY

R (a, b, c)

```
SELECT a, b as e  
FROM R  
ORDER BY e
```




ORDER BY

- Se puede indicar el número de columna en el ORDER BY

R (a, b, c)

```
SELECT a, c  
FROM R  
ORDER BY 2, 1
```



ORDER BY

- Se puede ordenar por una columna derivada (proveniente de un cálculo)

R (a, b, c)

```
SELECT a, b-c  
FROM R  
ORDER BY b-c
```



COCIENTE

En SQL no hay un operador que permita resolver directamente el cociente, tal como existe en A.R.

Existen varias formas de resolver el Cociente en SQL utilizando otros operadores. La forma mas recomendada es usando tres consultas anidadas vinculadas entre si mediante NOT EXISTS.



COCIENTE

Ejemplo:

ALUMNO (legajo, nom, ape, email, telefono)

MATERIA (cod_mat, nombre, año_de_la_carrera)

CURSA (legajo, cod_mat)

Listar los legajos y apellidos de los Alumnos que cursan **todas** las materias de cuarto año.



COCIENTE

Solución:

Listar los legajos y apellidos de los Alumnos que cursan **todas** las materias de cuarto año.

```
SELECT a.legajo, a.apellido
FROM Alumno a
WHERE NOT EXISTS ( SELECT *
                    FROM Materia m
                    WHERE año_de_la_carrera = 4
                    AND NOT EXISTS ( SELECT *
                                    FROM Cursa c
                                    WHERE c.legajo = a.legajo
                                    AND c.cod_mat = m.cod_mat ))
```



COCIENTE

Solución:

Listar los legajos y apellidos de los Alumnos que cursan **todas** las materias de cuarto año.

```
SELECT a.legajo, a.apellido
FROM Alumno a
WHERE NOT EXISTS ( SELECT *
                    FROM Materia m
                    WHERE año_de_la_carrera = 4
                    AND NOT EXISTS ( SELECT *
                                    FROM Cursa c
                                    WHERE c.legajo = a.legajo
                                    AND c.cod_mat = m.cod_mat ))
```

Alumnos tales que
NO EXISTE una Materia de 4to año
que NO cursen



COCIENTE

Hay otras dos formas de resolverlo:

1) Contar cuantas materias hay en 4to año y luego encontrar a los alumnos que cursan esa misma cantidad de materias de 4to año.

2) Resolverlo el cociente con operadores básico de AR y luego traducir esa expresión a SQL.

Todas las operaciones básicas que vimos en AR se pueden traducir al SQL.

$\text{Mat4to} \leftarrow \sigma_{\text{año_de_la_carrera}=4} (\text{Materia})$

$\text{CursanTodas} \leftarrow \pi_{\text{legajo}}(\text{Cursa}) - \pi_{\text{legajo}} ((\pi_{\text{legajo}}(\text{Cursa}) \times \text{Mat4to}) - \text{Cursa})$

$\text{Resultado} \leftarrow \pi_{\text{legajo}, \text{apellido}} (\text{CursanTodas} \mid \times \mid \text{Alumno})$



COCIENTE

Hay otras dos formas de resolverlo:

1) Contar cuantas materias hay en 4to año y luego encontrar a los alumnos que cursan esa misma cantidad de materias de 4to año.

2) Resolverlo el cociente con operadores básico de AR y luego traducir esa expresión a SQL.

Todas las operaciones básicas que vimos en AR se pueden traducir al SQL.

$\text{Mat4to} \leftarrow \sigma_{\text{año_de_la_carrera}=4} (\text{Materia})$

$\text{CursanTodas} \leftarrow \pi_{\text{legajo}(\text{Alumno})} - \pi_{\text{legajo} ((\pi_{\text{legajo}(\text{Alumno}) \times \text{Mat4to}) - \text{Cursa})$

$\text{Resultado} \leftarrow \pi_{\text{legajo, apellido} (\text{CursanTodas} \mid \times \mid \text{Alumno})$



Vistas

Las vistas son Consultas SQL almacenadas en la base de datos con un nombre.

A diferencia de las Tablas, las vistas no contienen información, sino que simplemente devuelven el resultado de la consulta la cual se ejecuta cada vez que la vista es invocada.

Las vistas se utilizan principalmente para:

1. Almacenar consultas que utilizamos con frecuencia
2. Para restringir permisos sobre ciertos datos (por ejemplo: tabla de empleados sin la columna salario)



Creación de una vista

CREATE VIEW nombre_vista [(nombre de columnas)] AS

Consulta SQL



Creación de una vista

Ejemplo:

EMPLEADO (legajo, nom, ape, fecha_ingreso, salario, cod_depto)
DEPARTAMENTO (cod_depto, descripcion, legajo_gerente)

```
CREATE VIEW Depto AS  
  SELECT descripción, count(*) as cant_empleados, g.ape as gerente  
  FROM Empleado e, Departamento d, Empleado g  
  WHERE e.cod_depto=d.cod_depto  
  AND d.legajo_gerente=g.legajo  
  GROUP BY descripción, g.ape
```



Creación de una vista

Luego podemos consultar la vista como si fuese una tabla:

```
SELECT *  
FROM Depto  
WHERE cant_empleados > 10
```



Eliminación de una vista

`DROP VIEW nombre_vista`

Ejemplo:

`DROP VIEW Depto`

Si hay otras vistas que la usan, deja eliminarla igual, pero luego las otras vistas darán error cuando se quieran utilizar.



Actualización de los datos de una vista

Si un usuario quiere modificar, insertar o eliminar un dato de una tabla, lo mejor es que lo haga sobre la misma tabla.

Sin embargo, en algunos casos es posible hacerlo sobre la vista y el cambio se aplica automáticamente sobre la tabla a la que apuntan.

No todas las vistas son actualizables, es decir, permiten que se modifiquen los datos. Eso depende de cada motor de base de datos y versión del mismo.

En términos generales, podemos decir que si la vista es sobre una sola tabla y contiene su clave, casi seguro que permitirá que se modifiquen los datos.

Por el contrario, si la vista es una consulta agrupada, entonces casi seguro que no permitirá modificar sus datos.



Sentencias DML (Data Manipulation Language)

Permiten modificar los datos de las tablas.
Se pueden deshacer (admiten Roll Back)

- INSERT
- UPDATE
- DELETE

Sentencias DDL (Data Definition Language)

Permiten modificar la estructura de las tablas.
NO se pueden deshacer (NO admiten Roll Back)

- TRUNCATE
- DROP
- ALTER
- CREATE



INSERT

Permite insertar una o varias filas en una tabla.

Se puede utilizar de dos formas:

Forma 1 (inserta de a una fila):

```
INSERT [INTO] nombre_tabla [(lista de columnas)]  
VALUES (lista de valores)
```

Ejemplo:

```
INSERT INTO Empleado (legajo, nombre, apellido)  
VALUES (10,'Jorge', 'Varela')
```




INSERT

Permite insertar una o varias filas en una tabla.

Se puede utilizar de dos formas:


Forma 1 (inserta de a una fila):

```
INSERT [INTO] nombre_tabla [(lista de columnas)]  
VALUES (lista de valores)
```

Si se omite, se supone que son todas las columnas de la tabla

Ejemplo:

```
INSERT INTO Empleado (legajo, nombre, apellido)  
VALUES (10,'Jorge', 'Varela')
```





INSERT

Forma 2 (inserta multiples filas):
Inserta en una tabla el resultado de una consulta.

```
INSERT [INTO] nombre_tabla [(lista de columnas)]  
  SELECT lista de columnas  
  FROM nombre_tabla
```

Ejemplo:
INSERT INTO Empleado_backup
SELECT *
FROM Empleado



DELETE


Permite eliminar una o varias filas en una tabla.

```
DELETE [FROM] nombre_tabla  
[WHERE condición]
```

Ejemplo:

```
DELETE Empleado  
WHERE cod_depto IN (5,4)
```

Puede dar error si se intenta
eliminar una fila referenciada por
una Foreign Key de otra tabla.





UPDATE

Permite modificar los valores de una o varias columnas de una o varias filas de una tabla.

```
UPDATE nombre_tabla  
SET columna1 = <nuevo valor>,  
    columna2 = <nuevo valor>,  
    ...  
    columnaN = <nuevo valor>  
[WHERE condición]
```

Ejemplo:

```
UPDATE Empleado  
SET    telefono = '555-1234', salario = 50000  
WHERE  legajo = 23
```



TRUNCATE

Permite eliminar TODAS las filas de una tabla.

```
TRUNCATE TABLE nombre_table
```

Ejemplo:

```
TRUNCATE TABLE Empleado
```



TRUNCATE

Es similar al DELETE, ya que ambas sentencias eliminan filas, pero tiene 3 diferencias:

1. El TRUNCATE elimina TODAS las filas de la tabla, no puede eliminar solo algunas
2. No tiene posibilidad de deshacerse, ya que no permite Roll Back.
Igualmente esto está cambiando y algunas versiones nuevas de algunos motores están comenzando a permitirlo.
3. Libera el espacio físico que ocupan las filas. El delete hace un borrado lógico y no libera el espacio.
Si vemos cuanto espacio ocupa una tabla, luego hacemos un DELETE y eliminamos la mitad de sus filas y por ultimo volvemos a ver cuanto espacio ocupa la tabla, veremos que sigue ocupando lo mismo. Ya que no libera el espacio de las filas eliminadas. El TRUNCATE si lo hace.