Universidad Nacional de La Matanza

Catedra de Base de Datos

Clase Teórica de SQL – Parte 2

SQL Clase 2

Consultas:

Para hacer una consulta con SQL se utiliza la sentencia SELECT.

Forma genérica:

```
SELECT [distinct] columna(s)
FROM   tabla(s)
WHERE  condicion(es)
GROUP BY columna(s)
HAVING condicion(es)
ORDER BY columna(s)
```

Como resultado, devuelve SIEMPRE una TABLA.

SQL Clase 2

Ejemplos:

```
SELECT * FROM Empleado

SELECT mail FROM Empleado WHERE legajo=1

SELECT mail,nya FROM Empleado
WHERE legajo>6 AND nac='AR'

SELECT apellido FROM Empleado
WHERE  legajo>=100 ORDER BY apellido

SELECT apellido FROM  Empleado
ORDER BY legajo desc

SELECT apellido FROM  Empleado
ORDER BY apellido desc, legajo asc
```

SQL Clase 2

WHERE

Las condiciones que se colocan en el WHERE, pueden utilizar cualquiera de los siguientes Operadores:

Operador	Ejemplo
=	legajo = 1
<>	nombre <> 'Juan'
>	fechaNac > '2015-01-01'
<	lejajo < 100
>=	fechaing >='2020-01-01'
<=	sueldo <= 20000
Like	nya like 'Perez%' / nya like '%perez%' / nya like '_perez'
between	Legajo between 20 and 50
IN (<lista de valores>)	legajo IN (10,20,30)
Is / is not	mail is null / mail is not null

SQL Clase 2

BETWEEN

Es un predicado de comparación.
Permite verificar si un valor se encuentra ENTRE otros dos valores.

Forma general:

Valor [NOT] BETWEEN valor1 AND valor2

Ejemplo:

```
SELECT *
FROM    Producto
WHERE precio BETWEEN 1000 AND 2000
```

SQL Clase 2

IS NULL

Se utiliza para evaluar si el valor de un atributo es NULL.
Recordemos que NULL o NULO es un valor especial, el cual representa No aplica o Desconocido.

Por lo tanto requiere una tratamiento especial.

Ejemplo 1:

```
SELECT *
FROM    Alumno
WHERE telefono IS NULL
```

Ejemplo 2:

```
SELECT *
FROM    Empleado
WHERE fecha_desvinculacion IS NOT NULL
```

SQL Clase 2

LIKE

Este operador permite buscar patrones en un texto

Utiliza dos comodines:

% representa cero o mas caracteres

_ representa exactamente un caracter

Forma general:

<Nombre de atributo> [NOT] LIKE <Patrón de búsqueda>

Ejemplos:

Si queremos buscar a los empleados cuyo apellido comienza con A, hacemos:

SELECT * FROM Empleado WHERE apellido LIKE 'A%'

Si queremos buscar a los empleados que se llaman Ezequiel o Exequiel, hacemos:

SELECT * FROM Empleado WHERE nombre LIKE 'E_equiel'

SOME/ANY y ALL

Estas sentencias permiten comparar un campo, utilizando cualquier operador, con un conjunto de valores devueltos por una subconsulta.

ANY y **SOME** actúan de igual modo y sólo existen ambos por compatibilidad.

Forma general:

campo <operador> SOME / ANY (subconsulta)
campo <operador> ALL (subconsulta)

Ejemplo 1: Listar los empleados de categoría 3 que cobran menos que algún empleado de categoría 2.

```
SELECT * FROM Empleado
WHERE categoria = 3
AND       salario < ANY ( SELECT salario FROM Empleado WHERE categoría = 2 )
```

Ejemplo 2: Listar el o los empleados que tiene el salario máximo:

```
SELECT * FROM Empleado
WHERE salario >= ALL ( SELECT salario FROM Empleado )
```

SQL Clase 2

IN

Permite verificar si un valor pertenece o no a un conjunto de valores.

Forma general:

<Nombre de atributo> [NOT] IN (<conjunto de valores>)

El conjunto de valores puede ser una lista de valores separados por coma o bien puede ser el resultado de una subconsulta.

Ejemplos:

Listar los proveedores de categoría 3, 5 y 6.

```
SELECT * FROM Proveedor WHERE categoria IN (3, 5, 6)
```

Listar los Clientes que viven en una Ciudad donde no hay ninguna Sucursal.

```
SELECT * FROM Cliente
WHERE cod_ciudad NOT IN ( SELECT cod_ciudad
                          FROM    Sucursal )
```

SQL Clase 2

EXISTS

Verifica la existencia de filas en una subconsulta.

Forma general:

[NOT] EXISTS (subconsulta)

Si la subconsulta devuelve una o mas filas, el EXISTS en verdadero.

Si la subconsulta no devuelve ninguna fila, el EXISTS en falso.

Ejemplo:

Listar los Departamentos donde NO trabaja algún Empleado.

```
SELECT *
FROM   Departamento d
WHERE NOT EXISTS ( SELECT *
                   FROM   Empleado e
                   WHERE e.cod_depto = d.cod_depto )
```

Por cada fila de Departamento, se evalua la subconsulta, considerando a d.cod_depto como si fuera una constante.

SQL Clase 2

UNION

UNION ALL

INTERSECT

EXCEPT

SQL Clase 2

UNION

Permite unir el resultado de dos consultas.

Elimina las filas duplicadas.

Forma general:

```
SELECT *
FROM    Tabla1
UNION
SELECT *
FROM    Tabla2
```

Para que el UNION pueda funcionar, ambas consultas **deben ser compatibles**, es decir, deben tener igual cantidad de columnas y los tipos de datos de ellas tienen que ser equivalentes.

El resultado toma los nombres de columna de la ;primer consulta.

UNION ALL

Es igual que el UNION pero no elimina las filas duplicadas.

Forma general:

```
SELECT  a, b
FROM    Tabla1
UNION ALL
SELECT  d, e
FROM    Tabla2
```

INTERSECT

Permite obtener las filas que son comunes a dos consultas (intersección).

Forma general:

```
SELECT  dni, nombre, apellido
FROM    Alumno
INTERSECT
SELECT  dni, nombre, apellido
FROM    Docente
```

Al igual que la UNION, para que el INTERSECT pueda funcionar, ambas consultas **deben ser compatibles.**

EXCEPT

Permite calcular la diferencia entre dos consultas.
Hace una resta entre la primer y la segunda consulta.
Devuelve las filas que están en la primer consulta y NO están en la segunda consulta.

Forma general:

```
SELECT  dni, nombre, apellido
FROM    Alumno
EXCEPT
SELECT  dni, nombre, apellido
FROM    Docente
```

Ambas consultas **deben ser compatibles.**

SQL Clase 2

JUNTAS

Es posible combinar dos o mas tablas, colocando ambas tablas en el FROM (separadas por coma) y luego indicando las condiciones de junta en el WHERE.

Forma general:

```
SELECT *
FROM    Tabla1, Tabla2
WHERE  <condición de junta>
```

Ejemplo:

```
SELECT  e.legajo, e.apellido, d.descripción AS Nombre_Depto
FROM     Empleado e, Departamento d
WHERE  d.cod_depto = d.cod_depto
```

Para mayor comodidad, se puede utilizar **Alias** en las tablas y en las columnas.

PRODUCTO CARTESIANO

Si colocamos dos tablas en el FROM pero no indicamos ninguna condición de junta en el WHERE, entonces la base de datos no va a saber como hacer la junta y devolverá una combinación de todas las filas de una tabla con todas las filas de la otra.
Es decir, hace TODOS contra TODOS.
Eso se llama producto cartesiano.

Forma general:

```
SELECT *
FROM    Tabla1, Tabla2
```

Ejemplo:

```
SELECT *
FROM Empleado, Departamento
```

SQL Clase 2

JUNTAS ANSI:

NATURAL JOIN

INNER JOIN

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

NATURAL JOIN

Realiza una junta “automática” mediante las columnas de igual nombre.

Ejemplo:

```
SELECT *
FROM  Empleado NATURAL JOIN Departamento
WHERE salario > 100000
```

No se usa casi nunca. En muchos motores de base de datos ni siquiera está implementado.

SQL Clase 2

[INNER] JOIN

Es una junta con condición.

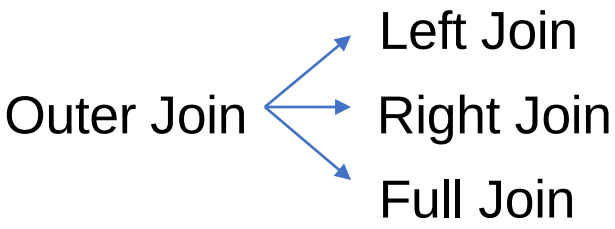
Ejemplo: Listar los Empleados del departamento de Sistemas.

```
SELECT *
FROM  Empleado e INNER JOIN Departamento d
          ON e.cod_depto = d.cod_depto
WHERE d.descripción = 'Sistemas'
```

La palabra INNER es opcional.

OUTER JOIN

Hay tres tipos de outer join



LEFT [OUTER] JOIN

Preserva todas las filas de la tabla izquierda por mas que no tengan ninguna fila para hacer junta en la tabla derecha.

Ejemplo:

R

A	B
a1	b1
a2	b2

S

C	D
b1	d1
b3	d2

```
SELECT *
FROM  R LEFT JOIN S ON R.B = S.C
```

A	B	C	D
a1	b1	b1	d1
a2	b2	null	null

RIGHT [OUTER] JOIN

Es igual que el Left Join pero preserva las filas de la tabla derecha.

Ejemplo:

R

A	B
a1	b1
a2	b2

S

C	D
b1	d1
b3	d2

```
SELECT *
FROM  R RIGHT JOIN S ON R.B = S.C
```

A	B	C	D
a1	b1	b1	d1
null	null	b3	d2

FULL [OUTER] JOIN

Es una combinación de las dos anteriores. Preserva las filas de ambas tablas.

Ejemplo:

R

A	B
a1	b1
a2	b2

S

C	D
b1	d1
b3	d2

```
SELECT *
FROM  R FULL JOIN S ON R.B = S.C
```

A	B	C	D
a1	b1	b1	d1
a2	b2	null	null
null	null	b3	d2

SQL Clase 2

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()

SQL Clase 2

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
```

SQL Clase 2

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 de Sistemas

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

SQL Clase 2

MAX() Devuelve el mayor valor de una columna
MIN () Devuelve el menor valor de una columna

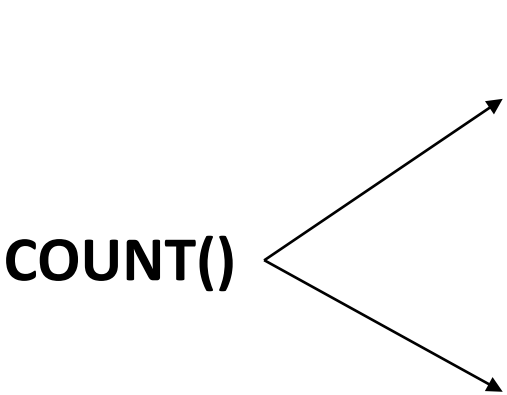
Ejemplo:

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
```

SQL Clase 2



COUNT(*)

Cuenta la cantidad de filas

COUNT(nombre columna)

Cuenta la cantidad de filas que no tienen valor nulo en esa columna.

SQL Clase 2

COUNT()

Ejemplo:

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

Cantidad de Departamentos

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

SQL Clase 2

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
```

SQL Clase 2

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
```

SQL Clase 2

GROUP BY

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

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

SQL Clase 2

GROUP BY

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

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

SQL Clase 2

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.

SQL Clase 2

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

SQL Clase 2

ORDER BY

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

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

SELECT ….

FROM ….

…..

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

Por defecto ordena en forma Ascendente

SQL Clase 2

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
```

SQL Clase 2

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
```

SQL Clase 2

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
```

SQL Clase 2

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
```

SQL Clase 2

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.

SQL Clase 2

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.

SQL Clase 2

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

SQL Clase 2

COCIENTE

Hay otra forma de resolverlo:

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

SQL Clase 2

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)

SQL Clase 2

Creación de una vista

CREATE VIEW nombre_vista [(nombre de columnas)] AS

Consulta SQL

SQL Clase 2

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
```

SQL Clase 2

Uso 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.

SQL Clase 2

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.