

LENGUAJES DE BASES DE DATOS: SQL

Francisco Moreno

Structured Query Language (SQL)

- Versión original: San José *Research Laboratory* de IBM, mediados de 1970
- Implementado por primera vez en un prototipo de IBM llamado System R
- El *American National Standard Institute* (ANSI) publicó el estándar SQL en 1986
- La última versión es **SQL:2023** (soporta JSON y grafos)
- Es el lenguaje estándar de los SGBD comerciales

Structured query Language (SQL)

Se divide en sublenguajes:

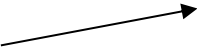

- **DDL**: Sublenguaje para crear, modificar y eliminar relaciones, crear y eliminar índices y vistas, permite especificar restricciones de integridad, entre otros aspectos
- **DCL**: Sublenguaje para crear usuarios y definir permisos de acceso
- **DML**: Sublenguaje de consulta (basado en álgebra y **cálculo relacional**), actualización, inserción y borrado de tuplas

Se ve posteriormente

Consultas: SELECT

- Su estructura esencial tiene tres elementos:
 - Una cláusula **SELECT**, que permite especificar los atributos que se desean en el resultado. Corresponde a la operación de **Proyección** del álgebra relacional.
 - Una cláusula **FROM**, que permite especificar las relaciones de la consulta. En su forma más simple corresponde a la operación **Producto Cartesiano** del álgebra.
 - Una cláusula **WHERE**, que permite especificar las condiciones de la consulta. Corresponde a la operación de **Restricción** del álgebra.

- En forma esquemática, se tiene:

SELECT DISTINCT a_1, a_2, \dots, a_n 
FROM R_1, R_2, \dots, R_m 
WHERE condición;

Lo anterior equivale a la siguiente expresión del álgebra:

$$\pi_{a_1, a_2, \dots, a_n} (\sigma_{\text{condición}} (R_1 \times R_2 \times \dots \times R_m))$$

- En la lista de atributos se puede colocar un asterisco (*), para representarlos a todos

- SQL **no elimina** tuplas repetidas* (si las hay). Para eliminarlas se usa **DISTINCT** después de **SELECT**.
 - Explícitamente se puede indicar que no se desea eliminar las posibles tuplas duplicadas, colocando **ALL**** después de **SELECT**.
-
- * Es decir, SQL **NO** es cerrado relacionalmente, por ejemplo, puede producir resultados con tuplas repetidas.
 - ** Es la opción predeterminada, rara vez se usa.

- Supóngase la relación EMPLEADO:

codigo	nombre	edad	depto
1	Jorge Campos	33	1
2	Enrique Muñoz	25	1
3	Esteban Paz	21	1
8	Jorge Arias	30	2
10	Juan Martínez	19	2
12	Anselmo Rodas	28	NULL

- Sea la consulta:

```
SELECT nombre, edad  
FROM empleado  
WHERE edad >= 28;
```

En SQL se prefieren
minúsculas para referirse a
las relaciones y a los
atributos

- La respuesta es:

nombre	edad
Jorge Campos	33
Jorge Arias	30
Anselmo Rodas	28

- Operadores de comparación:

= (igual), != o <> (diferente), > (mayor que), < (menor que), >= (mayor o igual que), <= (menor o igual que)

- Se pueden usar los conectores **AND**, **OR**, **NOT**:

Ej: **SELECT ***
FROM empleado
WHERE edad < 28 AND depto = 1;

- Se puede usar el operador **BETWEEN**, para especificar un rango de valores, por ejemplo:

Ej: **SELECT ***
FROM empleado
WHERE edad BETWEEN 18 AND 30;

BETWEEN se puede expresar por medio de **>=** y **<=**

- Se puede indicar una lista de valores con el operador **IN**:

```
SELECT *  
FROM empleado  
WHERE nombre IN ('Jorge Campos', 'Esteban Paz');
```

En vez de ser una lista estática de valores también se puede especificar una consulta, ver luego.



La condición anterior equivale a:

Ojo: ¿Qué pasaría si se colocase un **AND** acá?

```
nombre = 'Jorge Campos' OR nombre = 'Esteban Paz'
```

- Tanto **IN** como **BETWEEN** se pueden negar con **NOT**

- Las cadenas de caracteres se pueden comparar con (=) y diferente (<>, !=)*
- También se puede usar el operador LIKE para expresar comparaciones de cadenas de caracteres más complejas:
 - El carácter % reemplaza cualquier subcadena
 - El carácter _ reemplaza un carácter
- Ejemplos:
 - atributo LIKE 'amer%' : cadenas que comiencen por 'amer'
 - atributo LIKE '%eri%' : cadenas que contengan 'eri'
 - atributo LIKE '____%' : cadenas que tengan al menos tres caracteres

*También se pueden comparar cadenas con >, <, etc.; según el código ASCII.

- Operador de concatenación: ||

Operador de renombrado, en algunos SGBD no es necesario usarlo

```
SELECT nombre || ' ' || edad AS nomyedad  
FROM empleado;
```

Un espacio

nomyedad	
Jorge Campos	33
Enrique Muñoz	25
Esteban Paz	21
Jorge Arias	30
Juan Martínez	19
Anselmo Rodas	28

En algunos SGBD se debe usar la función TO_CHAR si se va a concatenar con un NUMBER

- Para comparar con nulos: `IS NULL` e `IS NOT NULL`:

```
SELECT * FROM empleado WHERE depto IS NULL;
```

- Para cambiar un nulo por un valor: `NVL` (en Oracle), `COALESCE`:

```
SELECT codigo, COALESCE(depto, -1) FROM empleado;
```

La cláusula WHEN

```
SELECT sn,  
       (CASE WHEN situacion < 20 THEN 'Malo'  
             WHEN situacion < 30 THEN 'Regular'  
             ELSE 'Bueno'  
       END  
       ) AS tipoprov  
FROM s;
```

Ejercicio: Quitar la línea `ELSE 'Bueno'` y ver que ocurre

- *Joins*:
 - Mediante la cláusula `INNER JOIN ... ON`
 - Se puede usar cualquier operador de comparación diferente al de igualdad produciendo *theta joins*
 - La forma “clásica” para hacer un *join* es comparar los atributos de *join* en la cláusula `WHERE`
 - Otra forma es usar la cláusula `NATURAL JOIN`

Sean las relaciones:

EMPLEADO

codigo	nombre	edad	depto
1	Jorge Campos	33	1
2	Enrique Muñoz	25	1
3	Esteban Paz	21	1
8	Jorge Arias	30	2
10	Juan Martínez	19	2
12	Anselmo Rodas	28	6

DEPARTAMENTO

depto	descripcion
1	Administración
2	Producción
3	Ventas
4	Finanzas

- Ejemplos de *joins*:

a) `SELECT *`

```
FROM empleado INNER JOIN departamento ON  
empleado.depto = departamento.depto;
```

b) `SELECT *`

```
FROM empleado, departamento  
WHERE empleado.depto = departamento.depto;
```

c) Nótese que la siguiente consulta imprime **solo una vez** el atributo de *join*:

`SELECT *`

```
FROM empleado NATURAL JOIN departamento;
```



Equivale a `EMPLEADO ⋈ DEPARTAMENTO`

- Los **INNER JOINs** se pueden anidar para lograr una sucesión de *joins*. Además los atributos de *join* no se tienen que llamar igual entre las tablas:

```
SELECT *  
FROM (t1 INNER JOIN t2 ON t1.a = t2.b)  
      INNER JOIN t3 ON t1.b = t3.c;
```

- Se puede usar la cláusula **AS** para especificar alias para las tablas.

```
SELECT e.codigo, e.nombre, d.*  
FROM empleado AS e, departamento AS d  
WHERE e.depto = d.depto;
```

Nota: **AS** hace parte del SQL estándar; sin embargo, algunos SGBD como **Oracle no lo soportan para dar alias a las tablas (en Oracle se omite)**.

AS también sirve para renombrar atributos (en este sentido **SÍ** funciona en Oracle, pero su uso es opcional).

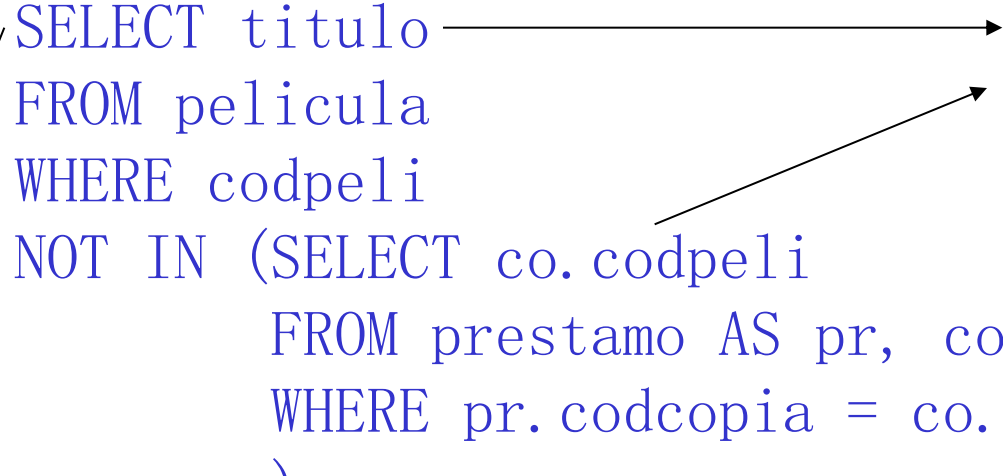
- Ejemplo: Sean las relaciones:
 - SOCIO (codsocio, nombre, direccion, telefono)
 - PELICULA (codpeli, titulo, genero)
 - COPIA (codcopia, codpeli)
 - PRESTAMO (codsocio, codcopia, fecha_pres, fecha_dev)

Nota: Aquí se han subrayado los atributos que forman la CP de cada relación.

- Ejemplo de una **subconsulta**.

Ej: Mostrar el título de las películas que nunca se han prestado:

```
SELECT titulo  
FROM pelicula  
WHERE codpeli  
NOT IN (SELECT co.codpeli  
        FROM prestamo AS pr, copia AS co  
        WHERE pr.codcopia = co.codcopia  
        );
```



¿Será necesario usar
acá **DISTINCT**?

¿Sería recomendable
incluir codpeli acá?

Estas funciones **hacen caso omiso de los nulos** pero consideran valores repetidos.

- Se pueden usar funciones de agregados:
 - SUM(atributo): Sumatoria de los valores del atributo.
 - MAX(atributo): Valor máximo del atributo.
 - MIN(atributo): Valor mínimo del atributo.
 - AVG(atributo): Valor promedio del atributo.
 - COUNT(atributo o *): Conteo de tuplas.
- Se puede usar GROUP BY con estas funciones para consolidar por grupos (operador G del álgebra)
- Se puede usar HAVING para establecer condiciones para los grupos (HAVING es para los grupos lo que el WHERE es para las tuplas)

- Ej: Obtener el número de veces que se ha prestado la película más prestada.

```
SELECT MAX(numero_veces) AS maxnumvec
FROM
(SELECT COUNT(*) AS numero_veces
 FROM prestamo AS pr, copia AS co
 WHERE pr.codcopia = co.codcopia
 GROUP BY co.codpeli
);
```

- En esta consulta se puede evitar, en algunos SGBD, el **SELECT** externo así:

```
SELECT MAX(COUNT(*)) AS maxnumvec  
FROM prestamo pr, copia co  
WHERE pr.codcopia = co.codcopia  
GROUP BY co.codpeli;
```


- En lo posible evite SELECTs innecesarios:

```
SELECT codsocio  
FROM (SELECT * FROM socio);
```

Esto es simplemente:

```
SELECT codsocio  
FROM socio;
```

Lamentablemente, el uso de consultas que son más complejas de lo necesario **es muy común** y esto puede **afectar negativamente** el rendimiento de las aplicaciones