

# ***Structured Query Language***

## ***Data Manipulation Language***

### **Administración de Bases de Datos**



Departamento de  
electrónica e informática

*Carlos Juan Martín Pérez*

# Structured Query Language

## Data Manipulation Language

**Tablas de Composición** (*joined tables*): Permite especificar, en la cláusula **FROM**, una relación que sea el resultado de una operación de **reunión/composición** (**JOIN**) entre dos tablas. **Formato:**  
**SELECT... FROM (<tabla1> JOIN <tabla2> ON <condic.>)**

**Composición con una Composición:** Una tabla de un **JOIN**, puede ser un **JOIN**: **((<t1> JOIN <t2> ON <cond1>) JOIN <t3> ON <cond2>)**

**Composición Natural:** Si los atributos se llaman **igual**, puede usarse **NATURAL JOIN** y eliminar la **condición** (no es habitual).

Los atributos con igual nombre son igualados y aparecen sólo una vez en el resultado. Si no se llaman igual puede usarse también **NATURAL JOIN**, cambiando los nombres de los atributos: **<t1> NATURAL JOIN (<t2> AS <nuevat2> (<nuevos nombres>))**

**Tipos de Composición:** SQL permite las expresiones: (**OUTER** es opcional)

**JOIN / INNER JOIN / NATURAL JOIN**

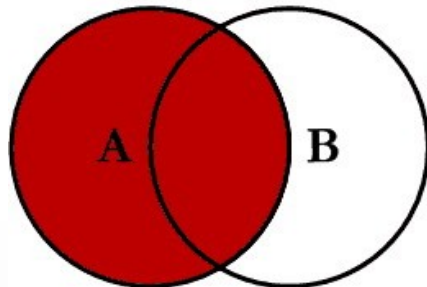
**LEFT OUTER JOIN / RIGHT OUTER JOIN**

**FULL OUTER JOIN**

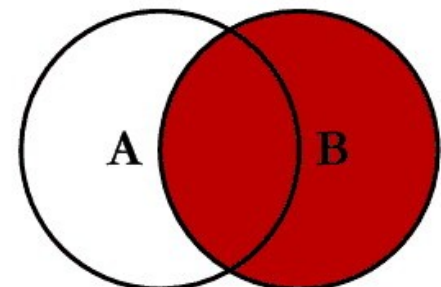
# Structured Query Language

## Data Manipulation Language

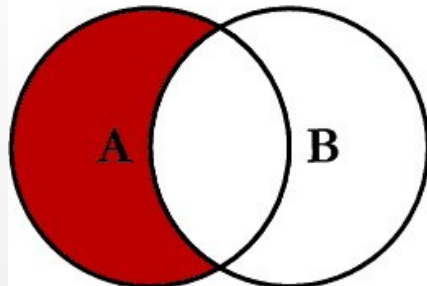
### SQL JOINS



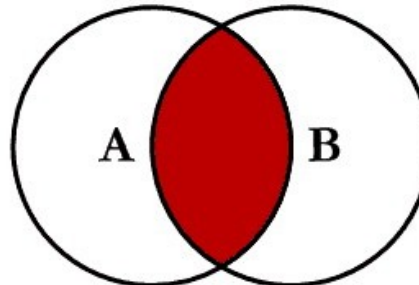
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



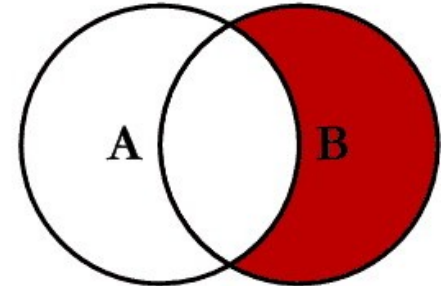
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



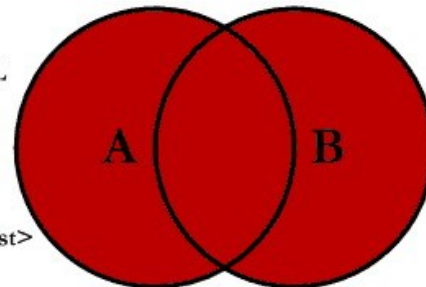
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



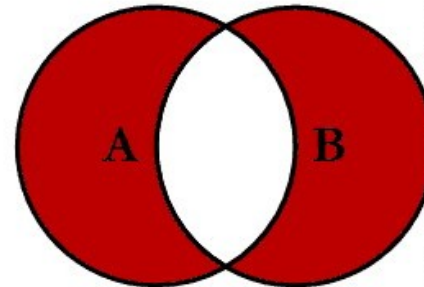
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

# Structured Query Language

## Data Manipulation Language

“Ids de **Suministradores** que Suministren una Pieza de 15 gramos”:

```
SELECT id_s
FROM (Pieza P JOIN Suministro SP ON P.id_p=SP.pieza_id_p)
WHERE Peso=15;
```

≡

```
SELECT id_s
FROM Pieza P, Suministro SP
WHERE P.id_p=SP.pieza_id_p AND Peso=15;
```

“Nombres de **Piezas** suministradas desde Soyapango”:

```
SELECT nombre_p
FROM ((Suministro SP JOIN Pieza P ON P.id_p=SP.pieza_id_p)
      JOIN Suministrador S ON S.id_s=SP.suministrador_id_s)
WHERE Ciudad='Soyapango';
```

“Nombres de los **empleados** y sus **supervisores** (si los tienen)”:

```
SELECT E1.Nombre Empleado, E2.Nombre Supervisor
FROM (Empleado E1 LEFT OUTER JOIN Empleado E2
      ON E1.DUI_Supervisor=E2.DUI);
```

# Structured Query Language

## Data Manipulation Language

**Subconsulta o Consulta anidada** (*nested query*): Sentencia SELECT incluida dentro de otra SELECT (consulta externa, *outer query*).

Una **subconsulta** genera un **multiconjunto** de valores: Podemos ver si un valor **pertenece** al multiconjunto (**IN**), si es mayor, igual, etc. que **todos** (**ALL**) o **alguno** (**ANY**) de los valores del multiconjunto:

```
SELECT * FROM Pieza, Suministrador
WHERE (id_p,id_s) IN (SELECT * FROM Suministro);
```

```
SELECT * FROM Pieza
WHERE id_p IN (
    SELECT id_p FROM Suministro SP, Suministrador S
    WHERE SP.id_s=S.id_s AND Ciudad='Soyapango')
OR id_p IN (
    SELECT id_p FROM Pieza
    WHERE Peso IS NOT NULL AND Cantidad IN (15,20));
```

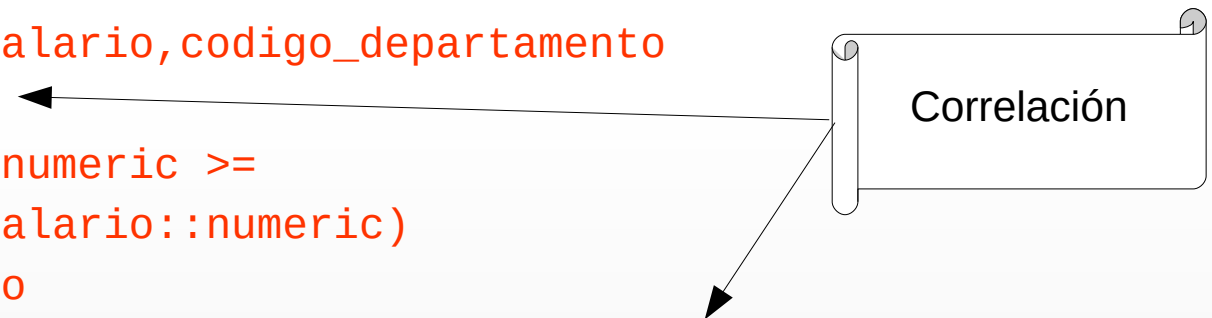
```
SELECT Nombre FROM Empleado
WHERE Salario >= ALL (SELECT Salario FROM Empleado);
```

# Structured Query Language

## Data Manipulation Language

**Subconsulta o Consulta anidada correlacionada:** La consulta interna se evalúa una vez por cada fila de la consulta externa.

```
SELECT Nombre,salario,codigo_departamento
FROM Empleado E
WHERE salario::numeric >=
  (SELECT AVG(salario::numeric)
   FROM Empleado
   WHERE codigo_departamento = E.codigo_departamento)
ORDER BY codigo_departamento,salario;
```



```
SELECT Nombre,salario,codigo_departamento
FROM Empleado E
WHERE Salario >= ALL (
  SELECT Salario FROM Empleado
  WHERE codigo_departamento = E.codigo_departamento);
```

# Structured Query Language

## Data Manipulation Language

**Función [NOT] EXISTS:** Comprueba si una subconsulta (normalmente correlacionada) es o no **vacía** (si recupera o no alguna tupla).

### Ejemplos:

“Empleados tales que **NO existen** compañeros en el mismo Dpto.”:

```
SELECT Nombre FROM Empleado E
WHERE NOT EXISTS (
    SELECT * FROM Empleado
    WHERE codigo_departamento = E.codigo_departamento AND
          DUI<>E.DUI);
```

“Piezas de las que haya menos de 100 y **NO exista** actualmente ningún suministrador”:

```
SELECT * FROM Pieza P
WHERE Cantidad<100 AND NOT EXISTS
(SELECT * FROM Suministro SP WHERE SP.pieza_id_p=P.id_p);
```

“Números de suministradores para los que **NO existen** piezas que **NO** sean suministradas por ellos (es decir, suministradores que suministran todas las piezas)”

```
SELECT suministrador_id_s FROM Suministro SP
WHERE NOT EXISTS (
    SELECT * FROM Pieza P
    WHERE NOT EXISTS
        (SELECT * FROM Suministro SP2
         WHERE SP2.suministrador_id_s=SP1.suministrador_id_s AND
              SP2.pieza_id_p=P.id_p));
```

# Structured Query Language

## Data Manipulation Language

### Funciones de Grupo en Subconsultas (correlacionadas o no):

“Nombre de los suministradores que suministran más de 5 piezas”:

```
SELECT nombre_s FROM Suministrador S
WHERE 5 < (SELECT COUNT(*) FROM Suministro
          WHERE S.id_s=suministrador_id_s);
```

“Nombre de las Piezas que pesan más que la media”:

```
SELECT nombre_p FROM Pieza
WHERE Peso > (SELECT AVG(Peso) FROM Pieza);
```

“Nombre de la pieza o piezas que pesen más”:

```
SELECT nombre_p FROM Pieza
WHERE Peso = (SELECT MAX(Peso) FROM Pieza);
```

“Nombre de piezas que son suministradas por varios suministradores”:

```
SELECT nombre_p FROM Pieza
WHERE 2 <= (SELECT COUNT(*) FROM
            Suministro WHERE pieza_id_p=Pieza.id_p);
```

“Nombre de piezas que provengan de 3 suministradores de Soya”:

```
SELECT nombre_p FROM Pieza P
WHERE 3 = (SELECT COUNT(*)
          FROM Suministros SP, Suministrador S
          WHERE pieza_id_p=P.id_p AND
                SP.suministrador_id_s=S.id_s AND
                Ciudad='Soyapango');
```



# Structured Query Language

## Data Manipulation Language

**Las Funciones de Grupo se pueden aplicar a subgrupos de entre las tuplas recuperadas** (no sólo al grupo formado por TODAS las tuplas recuperadas).

La **cláusula GROUP BY** seguida de una lista de atributos permite agrupar las tuplas en grupos que tengan los mismos valores en todos los atributos de esa lista.

En una consulta con **GROUP BY** todos los elementos seleccionados deben ser: Expresiones de la cláusula **GROUP BY**, Expresiones con funciones de grupo o Constantes.

“Para cada departamento, recuperar el número de empleados que tiene, su salario medio, y su mayor y menor salario”:

```
SELECT codigo_departamento, COUNT(*), AVG(Salario),  
       MAX(Salario), MIN(salario)  
FROM Empleado  
GROUP BY codigo_departamento;
```

“Para cada pieza, recuperar el número de suministradores que tiene”:

```
SELECT P.id_p, nombre_p, COUNT(*)  
FROM Pieza P, Suministro SP  
WHERE P.id_p=SP.pieza_id_p  
GROUP BY P.id_p, nombre_p;
```

## Structured Query Language

### Data Manipulation Language

**Cláusula HAVING:** Establece una condición sobre los grupos, para que sólo se recuperen los grupos que la cumplen:

“Para cada pieza con 2 o más suministradores, indicar cuántos son”:

```
SELECT id_p,nombre_p,COUNT(*) FROM Pieza,Suministro  
WHERE id_p=pieza_id_p  
GROUP BY id_p, nombre_p HAVING COUNT(*)>1;
```

“Departamentos y el número de sus empleados de aquellos que tienen más de 2 empleados y la media de su salario es mayor que 2200”:

```
SELECT codigo_departamento, COUNT(*) FROM Empleado  
GROUP BY codigo_departamento HAVING COUNT(*)>2 AND  
AVG(Salario::numeric)>2200;
```

“Recuperar los departamentos y el número de sus empleados que cobran más de 2500 de aquellos que tienen más de 2 empleados”:

```
SELECT codigo_departamento, COUNT(*) FROM Empleado  
WHERE Salario>2500 AND codigo_departamento IN  
(SELECT codigo_departamento FROM Empleado  
GROUP BY codigo_departamento HAVING COUNT(*)>2)  
GROUP BY codigo_departamento;
```

# Structured Query Language

## Data Manipulation Language

**Consultas recursivas:** realizan una consulta sobre una estructura recursiva (típicamente una FK que apunta a la PK de la misma tabla):

“¿Quiénes son los supervisores (directos e indirectos) de un empleado dado?”

Tabla temporal

**WITH RECURSIVE** supervisores(DUI, nombre, DUI\_supervisor)  
**AS (**

**SELECT** DUI, nombre, DUI\_supervisor  
**FROM** empleado **WHERE** DUI = '34531178-8'

Término no  
recursivo  
(resultado base)

**UNION ALL**

**SELECT** e.DUI, e.nombre, e.DUI\_supervisor  
**FROM** supervisores sup, empleado e  
**WHERE** e.DUI = sup.DUI\_supervisor

Término  
recursivo

)  
**SELECT** DUI, nombre, DUI\_supervisor  
**FROM** supervisores;

# Structured Query Language

## Data Manipulation Language

**SELECT** <Select\_list>

Expresiones a recuperar (atributos, funciones, operaciones...).

**FROM** <Table\_list>

Tablas necesarias para la consulta (incluyen tablas de reunión, subconsultas...).

[ **WHERE** <condición> ]

Condición de selección de tuplas, incluyendo condiciones de reunión.

[ **GROUP BY** <atributos\_para\_agrupar> ]

Atributos por los que agrupar el resultado (cada grupo tendrá los mismos valores en estos atributos). Las funciones de grupo o de agregación se aplicarán sobre cada uno de estos grupos. Se aplicarán sobre todas las tuplas si no existe esta cláusula.

[ **HAVING** <condición\_de\_grupo> ]

Condición sobre los grupos (no sobre las tuplas).

[ **ORDER BY** <atributos\_para\_ordenar> ]

Atributos por los que ordenar. El orden de estos atributos influye.

# Structured Query Language

## Data Manipulation Language

**VISTA:** Es una **tabla virtual** cuyas tuplas derivan de otras tablas (que pueden ser tablas base o también otras vistas).

Sus tuplas no se almacenan sino que se calculan a partir de las tablas de las que dependa.

Son útiles para usar, como si fueran tablas, consultas que se efectúan frecuentemente.

**CREATE [OR REPLACE] VIEW**

**<nombreV> [( <lista\_atrbs> )] AS ( <subconsulta> );**

Crea la vista **nombreV**, asociada a la subconsulta especificada.

La **lista de atributos** es el nombre de los atributos de la vista: Por defecto toma los nombres de los atributos de la subconsulta. Son necesarios si los atributos son calculados (funciones de grupo...).

**OR REPLACE:** Permite modificar una vista ya existente sin borrarla.

Existen Vistas modificables (materializables), prácticamente todos los SGBDR las soportan.

# Structured Query Language

## Data Manipulation Language

### Ejemplos:

```
CREATE OR REPLACE VIEW SumaNombres
AS (SELECT nombre_s, nombre_p
    FROM Suministro, Suministrador, Pieza
    WHERE suministrador_id_s=id_s AND
        pieza_id_p=id_p);

CREATE OR REPLACE VIEW Cantidad (nombre_p, NumSumin)
AS (SELECT nombre_p, COUNT(*)
    FROM Suministro, Pieza
    WHERE pieza_id_p=id_p
    GROUP BY nombre_p);
```

### Observaciones:

Las vistas pueden **consultarse como si fueran tablas**.

Una vista está **siempre actualizada** (*up to date*): Si se modifican las tablas de las que depende, la vista reflejará esos cambios.

Para **borrar una vista** que ya no es útil: **DROP VIEW** <nombreV>;