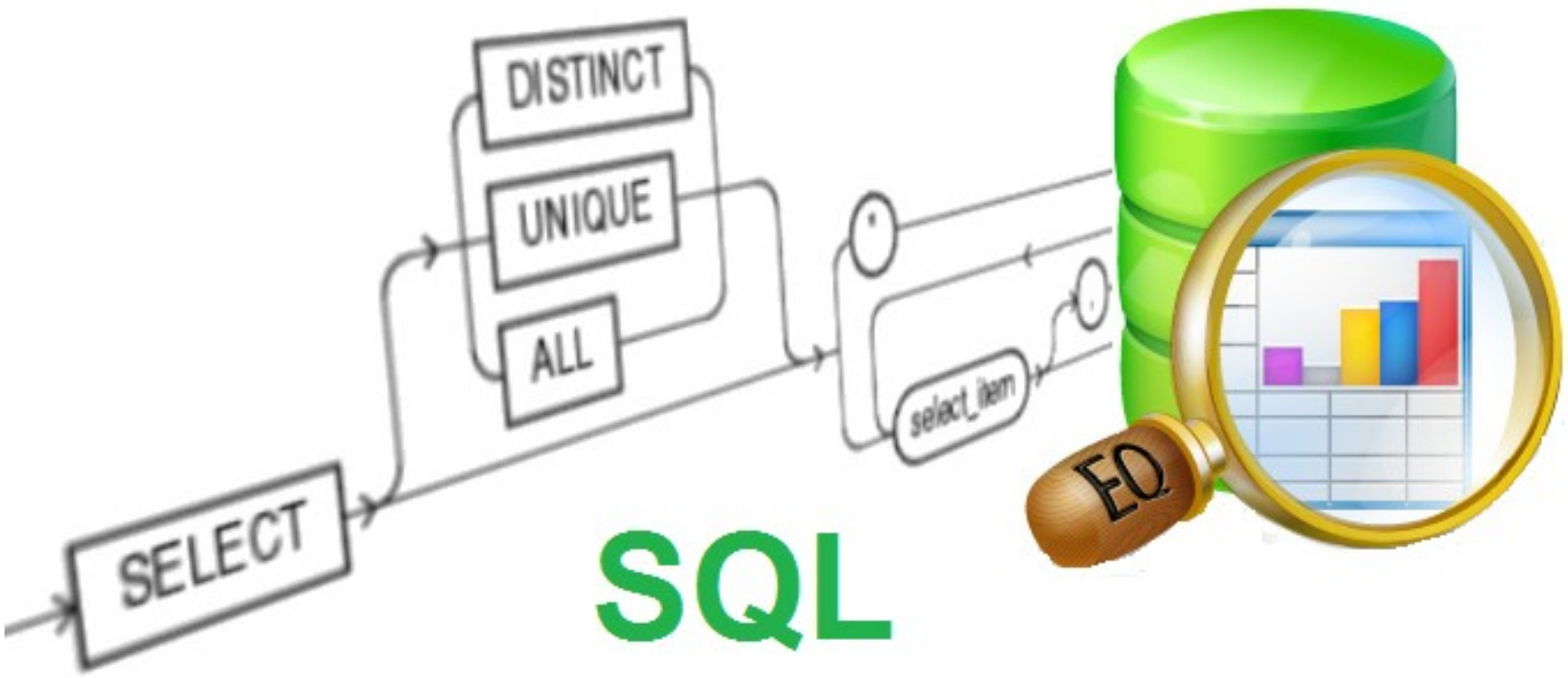
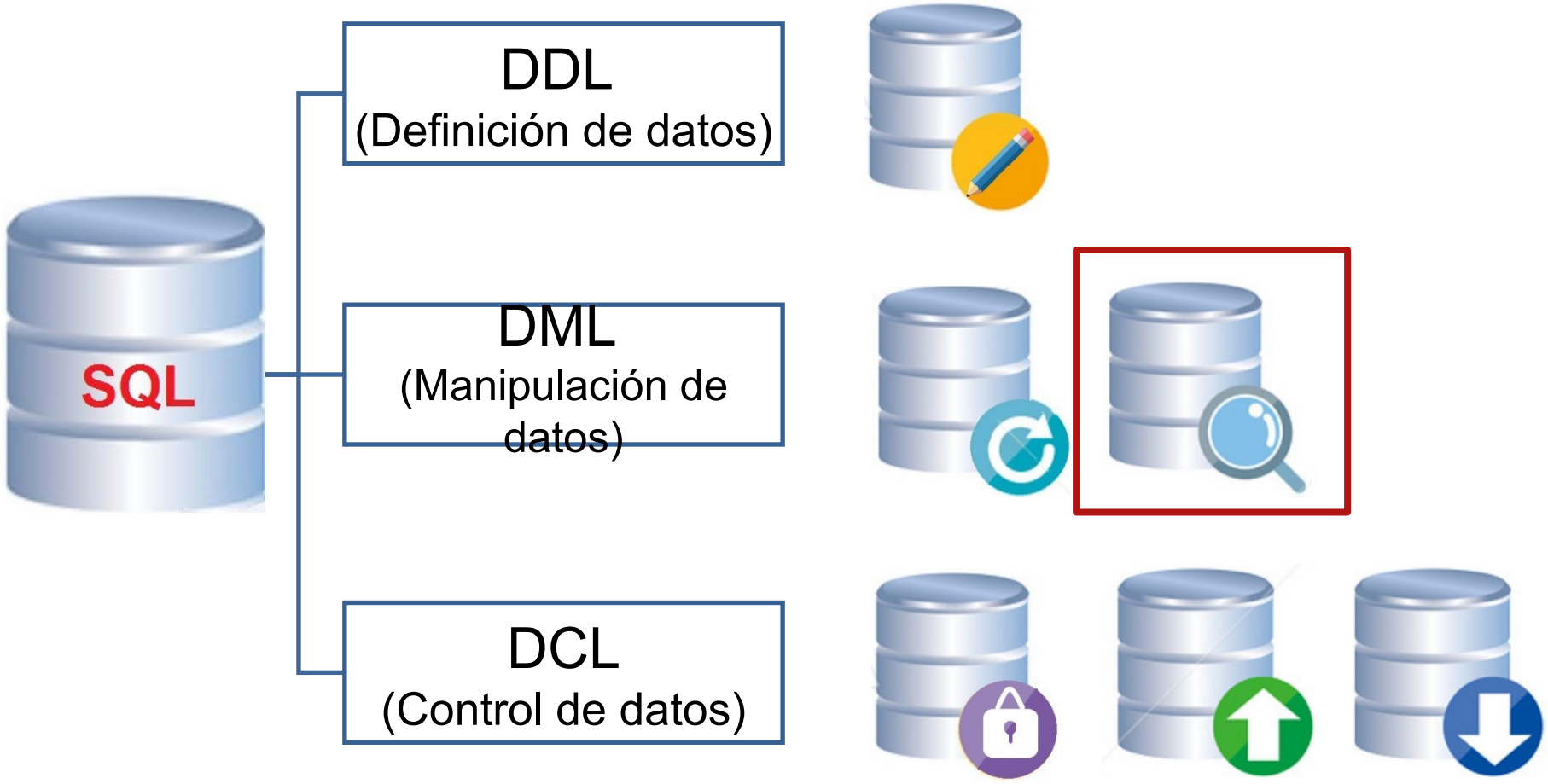
# Tema 4. Consultas



# Índice

# Introducción

Dentro de los sublenguajes SQL tenemos:



# Introducción

La cláusula SELECT permite realizar consultas y es el comando más versátil del lenguaje SQL.

El comando **SELECT** permite:

- Obtener datos de ciertas columnas de una tabla (proyección).
- Obtener registros (filas) de una tabla de acuerdo con ciertos criterios (selección).
- Mezclar datos de tablas diferentes (asociación, join).

# Consultas
## *Sintaxis básica*

**SELECT** [ALL | DISTINCT] [* | campo1, campo2, …]
**FROM** tabla1, tabla2, …
[**WHERE** condiciones];

Donde las opciones de Select son:

1. **ALL** muestra todos los registros, incluido los duplicados.
   Es la opción por defecto.
   **DISTINCT** elimina los duplicados
2. * muestra todos los campos
   Enumera los **campos** que se van a mostrar separados por comas.
   Se pueden **concatenar** cadenas con **concat(cadena1, cadena2)**
   **Expresiones aritméticas** (suma, resta, multiplicación y división).
   Puedes usar **alias** para los nombres de las columnas, con comillas dobles o el comando **AS**.
   Ejemplo: SELECT **concat(**nombre,' ', apellido1 **)** FROM PONENTE;
   Ejemplo: SELECT precio, precio*0.21 **AS** "IVA" FROM producto;

# Consultas
## *Sintaxis básica*

```
SELECT [ALL | DISTINCT] [* | campo1, campo2, …]
    FROM tabla1, tabla2, …
    [WHERE condiciones];
```

**FROM** *tabla*: especifica de qué tablas se obtiene la información buscada. Puedes usar **alias** a los nombres de las tablas, no es necesario comillas o el comando AS.
Ejemplo: SELECT * **FROM** SALA S;

**WHERE** *condicion*: expresa una condición que deben cumplir el/los registro/s de la consulta resultante (las filas).
Ejemplo: **SELECT** nombre, apellido1, apellido2
**FROM** ASISTENTE
**WHERE** sexo = 'H';

# Condiciones

## *Operadores de comparación*

| Operador | Descripción |
|---|---|
| = | Igualdad |
| !=, < >, ^= | Desigualdad. |
| < | Menos que |
| > | Mayor que. |
| <= | Menor o igual que. |
| >= | Mayor o igual que. |
| BETWEEN … AND … | permite definir rangos. |
| NOT BETWEEN …AND… | Fuera del rango. |

Comparan

En los textos, se compara en orden alfabético de la tabla de códigos.

La letra Ñ y las vocales acentuadas no se ordenan bien porque figuran con códigos más altos. Las mayúsculas van antes que las minúsculas (la ‘Z’ es menor que la ‘a’).

# Operadores Lógicos

| Operador | Descripción |
|---|---|
| ALL | TRUE si todos los valores de la subconsulta cumplen la condición |
| AND | TRUE si todas las condiciones separadas con AND son TRUE |
| ANY | TRUE si alguno de los valores cumplen la condición |
| BETWEEN | TRUE si esta entre el rango de comparación |
| EXISTS | TRUE si la subconsulta devuelve uno o más valores |
| IN | TRUE si el operando es igual a uno de una lista de expresiones |
| LIKE | TRUE si el operando coincide con un patrón |
| NOT | Muestra un registro si la/s condición/es NO es TRUE |
| OR | TRUE si alguna de las condiciones separadas por OR es TRUE |
| SOME | TRUE si alguno de los valores de la subconsulta cumple la condición |

# Operadores Lógicos

**Ejemplos de Operadores Lógicos**

*-- Personas entre 25 y 50 años*

**SELECT** nombre**,** apellidos

**FROM** PERSONAS

**WHERE** edad **>= 25 AND** edad **<= 50;**

*-- Personas de más de 60 y menos de 20*

**SELECT** nombre**,** apellidos

**FROM** PERSONAS

**WHERE** edad **> 60 OR** edad **< 20;**

# Operadores Lógicos

**Ejemplos de Operadores Lógicos**

- **BETTWEEN**

  *-- Personas entre 25 y 50 años*

  **SELECT** nombre**,** apellidos

  **FROM** PERSONAS

  **WHERE** edad **BETWEEN 25 AND 50;**

- **NOT BETTWEEN**

  *-- Personas no estén entre 25 y 50 años*

  **SELECT** nombre**,** apellidos

  **FROM** PERSONAS

  **WHERE** edad **NOT BETWEEN 25 AND 50;**

# Operadores Lógicos

**Ejemplos de Operadores Lógicos**

- **IN** *-- Selección de las piezas cuyo precio sea igual a 3, 5 u 8*

  **SELECT** tipo**,** modelo**,** precio

  **FROM** PIEZAS

  **WHERE** precio **IN ( 3,5,8 );**

- **IS NULL** *-- Devuelve el nombre y los apellidos de las personas que NO tienen teléfono*

  **SELECT** nombre**,** apellido1**,** apellido2

  **FROM** PERSONAS

  **WHERE** telefono **IS NULL**

# Operadores Lógicos

**Ejemplos de Operadores Lógicos**

- **LIKE:** busca cadenas con un patrón en campos de texto

  El patrón es una cadena que puede contener estos símbolos:

| Símbolo | Significado |
|---|---|
| _ | Sustituye a un carácter indeterminado |
| % | Sustituye un número indefinido de caracteres indeterminados. |

Sintaxis:
SELECT *column1, column2, ...*
FROM *table_name*
WHERE *columnN* **LIKE** *pattern*;

Ejemplo:
SELECT nombre, apellido1, apellido2
FROM PONENTE
WHERE codigo **LIKE** 'ESP**%**';

# Ordenación de registros

SELECT [ALL | DISTINCT] [* | campo1, campo2, …]
FROM tabla1, tabla2, …
[WHERE condicion]
**ORDER BY** campo1 [**ASC|DESC**], campo2 [**ASC|DESC**], … ;

**ORDER BY** campo: ordena los resultados por columnas (nº o letras o fecha)

**ASC/DESC** ascendente/descendente. Por defecto es Ascendente.

También se puede indicar el número de la columna.

Ejemplo: ordenamos por apellidos y nombre

```
SELECT nombre, apellido1, apellido2
FROM ASISTENTE
ORDER BY apellido1, apellido2, nombre;
```

Ejemplo: si queremos el resultado anterior ordenado por la Empresa:

```
SELECT nombre, apellido1, apellido2, empresa
FROM ASISTENTE
ORDER BY 4;
```

# Funciones

Todas las funciones reciben datos (**parámetros**) y devuelven un resultado (que depende de los parámetros enviados a la función.

```
NombreFunción [(parámetro1, [parámetro2, …)]
```

- Las funciones se pueden incluir en: SELECT, WHERE y ORDER BY.
- Puedes anidar funciones dentro de funciones.
- Existe una gran variedad de funciones para cada tipo de datos: numéricas, de cadena de caracteres, de manejo de fechas, …
- Oracle proporciona una tabla llamada DUAL con la que se permiten hacer pruebas. Esa tabla tiene un solo campo (llamado DUMMY) y una sola fila de modo que es posible hacer pruebas.
- Ejemplo: Calcula raíz de 5: SELECT SQRT(5) FROM DUAL;

# Funciones de caracteres*: CHAR(n) y VARCHAR(n)*

Las cadenas de caracteres se delimitan utilizando comillas simples.

Por ejemplo: ‘Hola’, ‘Una cadena’.

Además de los operadores de igualdad ( =, !=, ...) otras funciones útiles para trabajar con cadenas son:

**Concat (cad1, cad2)** : concatena dos cadenas.

**LENGTH(cad)**: devuelve la longitud de la cadena.

**LOWER(cad)**: convierte una cadena a minúsculas.

Ejemplo: SELECT LOWER('Hola'); -> hola

**UPPER(cad)**: convierte a mayúsculas.

Ejemplo: SELECT UPPER('Hola'); -> HOLA

# Funciones de caracteres*: CHAR(n) y VARCHAR(n)*

- **LTRIM(cad)** devuelve la cadena sin los espacios en blanco que pudiera contener al principio (en su parte izquierda). Ejemplo: SELECT **LTRIM**(' Hola'); -> Hola
- **RTRIM(cad)** devuelve la cadena sin los espacios en blanco que pudiera contener al final (en su parte derecha). Ejemplo: SELECT **RTRIM**('Hola '); -> Hola
- **TRIM(cad)** devuelve la cadena sin los espacios en blanco que pudiera contener al principio ni al final.

  Ejemplo: SELECT **TRIM**(' Hola '); -> Hola

  **TRIM(prefijo FROM cad)**: TRIM puede eliminar cualquier prefijo

  Ejemplo: SELECT **TRIM**('H' FROM 'Hola'); -> ola

# Funciones de caracteres*: CHAR(n) y VARCHAR(n)*

Otras funciones:

- **INSTR**
- **REPLACE**
- **LPAD**
- **RPAD**
- **REVERSE**

# Funciones numéricas

Funciones para redondear el número de decimales o redondear a números enteros:

- **ROUND(n, decimales)** redondea el número n al siguiente número con el número de decimales que se indican.
  Ejemplo: SELECT ROUND(2.5874, 2) FROM DUAL; -- Resultado: 2.59
- **TRUNCATE(m,n)** trunca al número m, la cantidad de decimales n.
  Si no existe n, se truncan todos los decimales.
  Si "n" es negativo, el número es truncado desde la parte entera.
  Ejemplos:
  SELECT TRUNCATE(127.4567, 2) FROM DUAL; -- Resultado: 127.45
  SELECT TRUNCATE(4572.5678) FROM DUAL; -- Resultado: 4572
  SELECT TRUNCATE(4572.5678, -2) FROM DUAL; -- Resultado: 4500
  SELECT TRUNCATE(4572.5678, -1) FROM DUAL; -- Resultado: 4570

# Funciones numéricas

- **ABS(n)** Calcula el de un número n.
  Ejemplo: SELECT ABS(-17) FROM DUAL; -- Resultado: 17
- **EXP(n)** Calcula $e^n$, es decir, el exponente en base e del número n.
  Ejemplo: SELECT EXP(2) FROM DUAL; -- Resultado: 7,38
- **MOD(m,n)**Calcula el resultante de dividir m entre n.
  Ejemplo: SELECT MOD(15, 2) FROM DUAL; --Resultado: 1
- **POWER(valor, exponente) o**
- **POW(v,e)** Eleva el valor al exponente indicado
  Ejemplo: SELECT POWER(4, 5) FROM DUAL; -- Resultado: 1024
- **SQRT(n)** Calcula la raíz cuadrada de n.
  Ejemplo: SELECT SQRT(25) FROM DUAL; --Resultado: 5
- **SIGN(n)** Si n es un valor positivo, retorna 1, si es negativo, devuelve -1 y 0 si es 0
  Ejemplo: SELECT SIGN(-23) FROM DUAL; – Resultado: -1

# Funciones de fechas

Las fechas se utilizan muchísimo en todas las bases de datos. Y suelen dar problemas por los formatos.

Tipos de datos para manejar fechas:

- **DATE**: se almacena una fecha concreta. Formato: YYYY-MM-DD
- **DATETIME o TIMESTAMP**: se almacena un instante de tiempo más concreto. Formato YYYY-MM-DD HH:MM:SS
- **TIME**: almacena la hora con el formato HH:MM:SS
- **YEAR**: almacena el año con el formato YYYY

Hay que tener en cuenta que a los valores de tipo fecha se les pueden sumar números y se entendería que esta suma es de días.

Si tiene decimales entonces se suman días, horas, minutos y segundos.

La diferencia entre dos fechas también obtiene un número de días.

# Funciones de fechas

Funciones para obtener la fecha y hora actual

| Función | Descripción |
|---|---|
| SYSDATE ()<br>NOW ()<br>CURRENT_TIMESTAMP ()<br>LOCALTIME()<br>LOCALTIMESTAMP() | Obtiene la fecha y hora actuales |
| CURDATE()<br>CURRENTDATE() | Obtiene la fecha y hora actuales en formato **TIMESTAMP** |

# Funciones de fechas

Funciones para calcular fechas:

| Función | Descripción |
|---|---|
| ADDDATE (date, INTERVAL unidad) | Suma un intervalo de tiempo a una fecha<br>SELECT ADDDATE('2022-01-01',<br>INTERVAL 31 DAY); |
| select YEAR(NOW());<br>select MONTH (NOW()) as mes;<br>select DAY(NOW()) as dia;<br>select TIME(NOW()) as hora;<br>Select LAST_DAY(NOW()); | Extraer parte de una fecha<br>Selecciona el año<br>Selecciona el mes<br>Selecciona el día<br>Selecciona la hora<br>Selecciona el ultimo dia del mes |
| DATE_FORMAT(fecha,formato);<br>Date_format(now(),'%Y/%M/%d')<br>Date_format(now(),'%Y-%M-%d %h:%i:%s %p');<br>Date_format(now(),'%W %d %M %Y');<br>Date_format(now(),'El año actual es %Y'); | Da formato a las fechas.<br>'2010/January/12'<br>'2010-January-12 12:34:29 AM'<br>'Tuesday 12 January 2010'<br>'El año actual es 2010' |

# Funciones de fechas

Códigos de las fechas

| Oracle TO_CHAR | | MySQL DATE_FORMAT |
|---|---|---|
| YYYY | Año con 4 digitos | %Y |
| YY | Año con 2 dígitos | %y |
| MONTH | Nombre del mes (January - December) | %M |
| MM | Mes (1 - 12) | %m |
| DD | Día del mes (1 - 31) | %d |
| HH24 | Hora (0 - 23) | %H |
| HH or HH12 | Hora (1 - 12) | %h |
| MI | Minuto (0 - 59) | %i |
| SS | Segundos (0 - 59) | %s |

EJEMPL

select D

select Date_format(now(),'%Y-%M-%d %h:%i:%s %p'); #'2022-January-12 12:34:29 AM'

select Date_format(now(),'%W %d %M %Y'); # 'Tuesday 12 January 2022'

# Conversiones implícitas

Se puede convertir datos automáticamente a fin de que la expresión final tenga sentido. En ese sentido son fáciles las conversiones de texto a número y viceversa.

Ejemplos:

- SELECT **5+'3'** FROM DUAL → *El resultado es 8*;
- SELECT **concat (5,'3')** FROM DUAL; → *El resultado es 53*

Pero en determinadas ocasiones querremos realizar conversiones explícitas. Para hacerlo utilizaremos las funciones que se detallan a continuación.

# Conversiones explícitas

Las más utilizadas son:

- CAST(expression AS type)
- CONVERT(expression, type)
- CONVERT(expr USING transcoding_name)

Ejemplos:

```
SELECT CAST('2022-01-01' AS DATE);
SELECT CAST(campo AS CHAR);
SELECT CAST(1 AS UNSIGNED) - 2.0;  -> -1.0
SELECT CONVERT('abc' USING utf8);
```

Valores de 'type':

- BINARY
- CHAR
- DATE
- DATETIME
- SIGNED {INTEGER}
- TIME
- UNSIGNED {INTEGER}

# Agrupaciones

La sentencia SELECT nos va a permitir obtener resúmenes, **consultas de resumen o Funciones Agregadas**. Estas consultas no corresponde con ningún valor de la tabla sino un total calculado sobre los datos de la tabla.

SINTAXIS:

```
SELECT funcion(campo), …
FROM tablas
WHERE condicion
GROUP BY campos
HAVING condición
ORDER BY campo;
```

- **GROUP BY** para agrupar filas y hacer subtotales
- **HAVING** para imponer una condición a las filas que agrupas

# Agrupaciones

## *Funciones de calculo de grupo*

Las funciones agregadas, **función** (campos), calculan un valor a partir de un campo de datos. Existen:

- **SUM:** suma de los valores. Sólo puede utilizarse con columnas cuyo tipo de dato sea número. El resultado será del mismo tipo aunque puede tener una precisión mayor.

  Ejemplo: SELECT SUM(precio) FROM CONFERENCIA;

- **COUNT:** cuenta los elementos de un campo. Puede contar hasta textos.

  COUNT(*), calcularemos el total de filas, incluyendo aquellas que contienen valores NULL.

  Ejemplos:

  SELECT COUNT(nombre) FROM PONENTE;

  SELECT COUNT(*) FROM PONENTE;

# Agrupaciones

## *Funciones de calculo de grupo*

- **MIN** : mínimo sin considerar los nulos (NULL). Incluye el nombre de un campo de una tabla, una constante o una función (pero no otras funciones agregadas de SQL).

  Ejemplo: SELECT MIN(precio) FROM CONFERENCIA;

- **MAX** máximo sin considerar los nulos (NULL). Incluye el nombre de un campo de una tabla, una constante o una función (pero no otras funciones agregadas de SQL).

  Ejemplo: SELECT MAX(precio) FROM CONFERENCIA;

- **AVG** media de los valores de un grupo, para ello se omiten los valores nulos (NULL).

  Se aplica a campos tipo número y el tipo de dato del resultado puede variar según las necesidades del sistema para representar el valor

# Consultas de varias tablas

Normalmente queremos información de varias tablas con algún campo en común. Los registros estarán relacionados con clave primaria y clave extranjera.

**¿Qué ocurre si combinamos dos o más tablas sin ninguna restricción?**

**Ejemplo**: SELECT tabla1.campo1, tabla1.campo2, tabla2.campo1, …
FROM tabla1, tabla2

El resultado será un **producto cartesiano**. El entre dos tablas da como resultado todas las combinaciones de todas las filas de esas dos tablas. Si tienes dos tablas de 10 filas cada una, el resultado tendrá 10x10 filas.

# Consultas de varias tablas

## *Producto cruzado o cartesiano de tablas: EJEMPLO*

Ejemplo: tabla de EMPLEADOS cuya clave principal es el DNI y otra tabla de TAREAS que se refiere a las tareas realizadas por los empleados. Suponemos que cada empleado realizará múltiples tareas, pero que cada tarea es realizada por un único empleado. Si el diseño está bien hecho, en la tabla de TAREAS aparecerá el DNI del empleado (como clave foránea) para saber qué empleado realizó la tarea.

| Empleados | | |
|---|---|---|
| **dni** | nombre_empleado | sueldo |
| 111111111A | Ana | 1200 |
| 222222222B | Bernardo | 1300 |

| Tareas | | |
|---|---|---|
| **cod_tarea** | descripcion | dni_empleado |
| 1 | Tarea 1 | 111111111A |
| 2 | Tarea 2 | 111111111A |
| 3 | Tarea 3 | 222222222B |
| 4 | Tarea 4 | 222222222B |

# Consultas de varias tablas

## *Producto cruzado o cartesiano de tablas: EJEMPLO*

Ejemplo:

**SELECT** cod_tarea**,** descripción_tarea**,** dni_empleado**,** nombre_empleado
**FROM** TAREAS**,** EMPLEADOS**;**

| cod_tarea | descripcion | dni_empleado | nombre_empleado |
|---|---|---|---|
| 1 | Tarea 1 | 111111111A | Ana |
| 2 | Tarea 2 | 111111111A | Ana |
| 3 | Tarea 3 | 222222222B | Ana |
| 4 | Tarea 4 | 222222222B | Ana |
| 1 | Tarea 1 | 111111111A | Bernardo |
| 2 | Tarea 2 | 111111111A | Bernardo |
| 3 | Tarea 3 | 222222222B | Bernardo |
| 4 | Tarea 4 | 222222222B | Bernardo |

# Consultas de varias tablas

## *Asociando tablas por igualdad: EJEMPLO*

Lo correcto, sería asociar las tareas con los empleados que la realizaron:

**SELECT** cod_tarea**,** descripción_tarea**,** dni_empleado**,** nombre_empleado
**FROM** TAREAS**,** EMPLEADOS
**WHERE** TAREAS.dni_empleado **=** EMPLEADOS.dni**;**

| cod_tarea | descripcion | dni_empleado | nombre_empleado |
|---|---|---|---|
| 1 | Tarea 1 | 111111111A | Ana |
| 2 | Tarea 2 | 111111111A | Ana |
| 3 | Tarea 3 | 222222222B | Bernardo |
| 4 | Tarea 4 | 222222222B | Bernardo |

# Consultas de varias tablas

## *Asociando tablas por igualdad : EJEMPLO*

La notación **tabla.columna** se utiliza para evitar la ambigüedad, ya que el mismo nombre de campo se puede repetir en ambas tablas.

Para evitar repetir continuamente el nombre de la tabla, se puede utilizar un alias de tabla.

**Ejemplo:** SELECT **T.**cod_tarea**, T.**descripción_tarea**, E.**dni**, E.**nombre
FROM TAREAS **T,** EMPLEADOS **E** WHERE **T.**dni_empleado **= E.**dni**;**

A la sintaxis WHERE se le pueden añadir condiciones sin más que encadenarlas con el operador AND.

**Ejemplo:** SELECT T.cod_tarea, T.descripción_tarea, E.dni, E.nombre
FROM TAREAS T, EMPLEADOS E
WHERE T.dni_empleado = E.dni **AND** E.nombre= 'Esther';

# Consultas de varias tablas

## *Relaciones sin igualdad*

A las relaciones descritas anteriormente se las llama relaciones en igualdad (equijoins), ya que las tablas se relacionan a través de campos que contienen valores iguales en dos tablas. A veces esto no ocurre, en las tablas.
Ejemplo:

| Empleados | | |
|---|---|---|
| **dni** | nombre | sueldo |
| 111111111A | Ana | 1200 |
| 222222222B | Bernardo | 1300 |
| 333333333C | Carlos | 1400 |

| Categorias | | |
|---|---|---|
| **categoria** | sueldo_minimo | sueldo_maximo |
| D | 6000 | 11999 |
| C | 12000 | 17999 |
| B | 18000 | 19999 |
| A | 20000 | 70000 |

# Consultas de varias tablas

## *Relaciones sin igualdad*

Ahora tenemos que averiguar la categoría a la que pertenece cada empleado, pero estas tablas poseen una relación que ya no es de igualdad.

La forma sería:

**SELECT** E.empleado, E.sueldo, C.categoria
**FROM** EMPLEADOS E, CATEGORÍAS C
**WHERE** E.sueldo **BETWEEN** C.sueldo_mínimo **AND** C.sueldo_máximo;

# Consultas de varias tablas
## *Combinación de tablas (JOIN)*

Existe otra forma más moderna e intuitiva de trabajar con varias tablas. Para ello se utiliza la clausula JOIN.

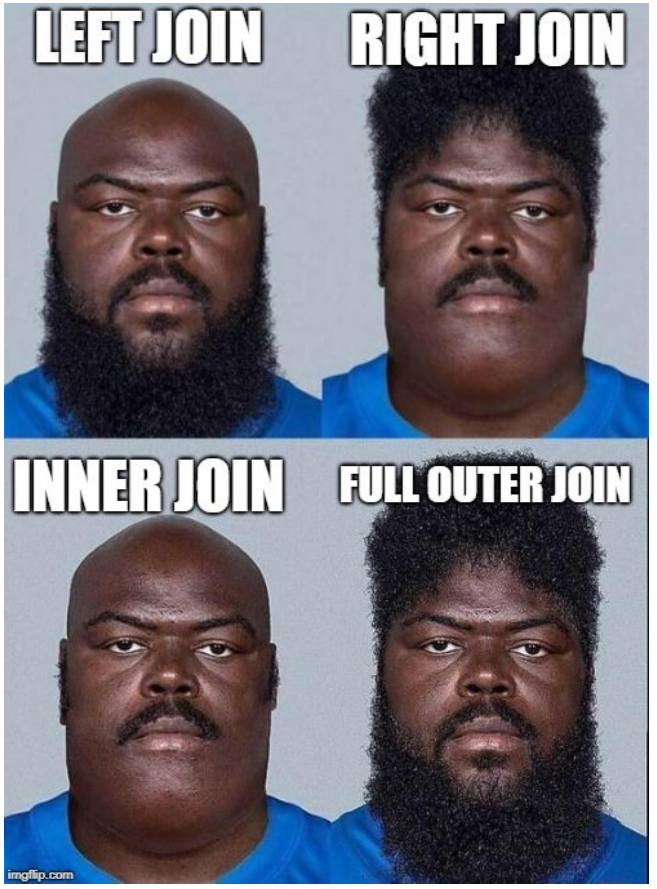
SINTAXIS:

```
SELECT tabla1.campo1, tabla1.campo2, tabla2.campo1, …
FROM tablas
[CROSS JOIN tabla2] |
[NATURAL JOIN tabla2] |
[JOIN tabla2 USING (columna) |
[JOIN tabla2 ON (tabla1.columna=tabla2.columna)] |
[LEFT | RIGTH | FULL OUTER JOIN tabla2 ON
(tabla1.columna=tabla2.columna)];
```

# Consultas de varias tablas
## *Combinación de tablas (JOIN)*

Existen diversas formas de combinar (JOIN) las tablas según la información que deseemos obtener. Los tipos de JOIN se clasifican en:

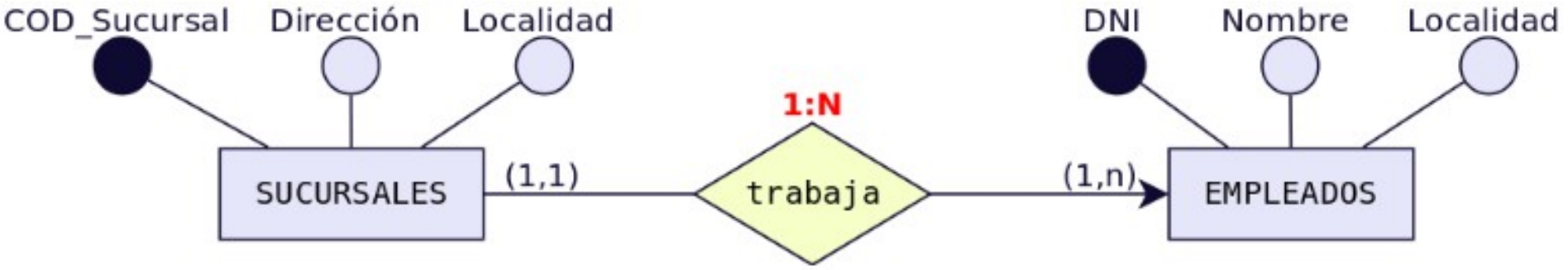
- INNER JOIN ( o simplemente JOIN): Combinación interna.
  - **JOIN**
  - SELF JOIN
  - NATURAL JOIN
- OUTER JOIN: Combinación externa.
  - LEFT OUTER JOIN (o **LEFT JOIN**)
  - RIGHT OUTER JOIN (o **RIGHT JOIN**)
  - **FULL OUTER JOIN** (o FULL JOIN)
- CROSS JOIN: Combinación cruzada.



# Consultas de varias tablas

## *Combinación de tablas (JOIN): Ejemplo*

Ejemplo: BD de un banco, con las tablas de empleados y sucursales.
En una sucursal trabajan varios empleados. Los empleados viven en una localidad y trabajan en una sucursal situada en la misma u otra localidad.




| Empleados | | | |
|---|---|---|---|
| **dni** | nombre | localidad | Cod_sucursal |
| 11111111A | Ana | ALMERÍA | 0001 |
| 22222222B | Bea | GRANADA | 0001 |
| 33333333C | Carlos | GRANADA | |
| 44444444D | David | JEREZ | 0003 |

| Sucursales | | |
|---|---|---|
| **Cod_sucursal** | direccion | localidad |
| 0001 | C/ ANCHA, 1 | ALMERÍA |
| 0002 | C/ NUEVA, 1 | GRANADA |
| 0003 | C/ CORTÉS, 3 | CADIZ |

# Consultas de varias tablas
## *Combinación INTERNA de tablas (JOIN)*

- **JOIN** o **INNER JOIN**: se utiliza para unir tablas en la que los nombres de columna no coinciden en ambas tablas o se necesita establecer asociaciones más complicadas.
  Ejemplo:
  **SELECT** E.*, S.LOCALIDAD **FROM** EMPLEADOS E
  **JOIN** SUCURSALES S **ON** E.COD_SUCURSAL = S.COD_SUCURSAL;

| dni | nombre | localidad | Cod_sucursal | localidad |
|---|---|---|---|---|
| 11111111A | Ana | ALMERÍA | 0001 | ALMERÍA |
| 22222222B | Bea | GRANADA | 0001 | ALMERÍA |
| 44444444D | David | JEREZ | 0003 | CADIZ |

# Consultas de varias tablas
## *Combinación INTERNA de tablas (JOIN)*

- **NATURAL JOIN**: detecta automáticamente las claves de unión, basándose en el nombre de la columna que coincide en ambas tablas. Por supuesto, se requerirá que las columnas de unión tengan el mismo nombre en cada tabla. Además, esta característica funcionará incluso si no están definidas las claves primarias o ajenas.
  Ejemplo:
  **SELECT * FROM** EMPLEADOS E **NATURAL JOIN** SUCURSALES S**;**

| Cod_sucursal | localidad | dni | nombre | direccion |
|---|---|---|---|---|
| 0001 | ALMERÍA | 11111111A | Ana | C/ ANCHA, 1 |

# Consultas de varias tablas

## *Combinación EXTERNA de tablas (JOIN)*

**LEFT OUTER JOIN:** es una composición externa izquierda, todas las filas de la tabla de la izquierda se devuelven, aunque no haya ninguna columna correspondiente en las tablas combinadas.

**RIGTH OUTER JOIN:** es una composición externa derecha, todas las filas de la tabla de la derecha se devuelven, aunque no haya ninguna columna correspondiente en las tablas combinadas.

**FULL OUTER JOIN**: es una composición externa en la que se devolverán todas las filas de los campos no relacionados de ambas tablas.

**CROSS JOIN**: creará un producto cartesiano de las filas de ambas tablas por lo que podemos olvidarnos de la cláusula WHERE.

**OUTER JOIN:** se puede eliminar el uso del signo (+) para composiciones externas utilizando un OUTER JOIN, de este modo resultará más fácil de entender.

# Consultas de varias tablas

## *Combinación EXTERNA de tablas (JOIN): Ejemplos*

**Ejemplo de LEFT JOIN**

```
SELECT E.*, S.LOCALIDAD
FROM EMPLEADOS E LEFT JOIN SUCURSALES S
ON E.COD_SUCURSAL = S.COD_SUCURSAL;
```

| dni | nombre | localidad | Cod_sucursal | localidad |
|---|---|---|---|---|
| 11111111A | Ana | ALMERÍA | 0001 | ALMERÍA |
| 22222222B | Bea | GRANADA | 0001 | ALMERÍA |
| 33333333C | Carlos | GRANADA | | |
| 44444444D | David | JEREZ | 0003 | CADIZ |

# Consultas de varias tablas

## *Combinación EXTERNA de tablas (JOIN): Ejemplos*

**Ejemplo de RIGHT JOIN**

```
SELECT E.DNI, E.NOMBRE, S.*
FROM EMPLEADOS E RIGHT JOIN SUCURSALES S
ON E.COD_SUCURSAL = S.COD_SUCURSAL;
```

| dni | nombre | localidad | Cod_sucursal | direccion | localidad |
|---|---|---|---|---|---|
| 11111111A | Ana | ALMERÍA | 0001 | C/ ANCHA, 1 | ALMERÍA |
| 22222222B | Bea | GRANADA | 0001 | C/ ANCHA, 1 | ALMERÍA |
| 44444444D | David | JEREZ | 0003 | C/ CORTÉS, 3 | CADIZ |
| | | | 0002 | C/ NUEVA, 1 | GRANADA |

# Consultas de varias tablas

## *Combinación EXTERNA de tablas (JOIN): Ejemplos*

**Ejemplo de FULL JOIN**

```
SELECT E.DNI, E.NOMBRE, S.COD_SUCURSAL, S.LOCALIDAD
FROM EMPLEADOS E FULL JOIN SUCURSALES S
ON E.COD_SUCURSAL = S.COD_SUCURSAL;
```

| dni | nombre | Cod_sucursal | localidad |
|---|---|---|---|
| 11111111A | Ana | 0001 | ALMERÍA |
| 22222222B | Bea | 0001 | ALMERÍA |
| | | 0002 | GRANADA |
| 44444444D | David | 0003 | CADIZ |
| 33333333C | Carlos | | |

# Consultas de varias tablas

## *Combinación EXTERNA de tablas (JOIN): Ejemplos*

**Ejemplo de CROSS JOIN**

**SELECT** E.DNI, E.NOMBRE, E.LOCALIDAD, S.COD_SUCURSAL, S.LOCALIDAD
**FROM** EMPLEADOS E **CROSS JOIN** SUCURSALES S;

| dni | nombre | localidad | Cod_sucursal | localidad |
|---|---|---|---|---|
| 11111111A | Ana | ALMERÍA | 0001 | ALMERÍA |
| 11111111A | Ana | ALMERÍA | 0002 | GRANADA |
| 11111111A | Ana | ALMERÍA | 0003 | CADIZ |
| 22222222B | Bea | GRANADA | 0001 | ALMERÍA |
| 22222222B | Bea | GRANADA | 0002 | GRANADA |
| 22222222B | Bea | GRANADA | 0003 | CADIZ |
| 33333333C | Carlos | GRANADA | 0001 | ALMERÍA |
| 33333333C | Carlos | GRANADA | 0002 | GRANADA |
| 33333333C | Carlos | GRANADA | 0003 | CADIZ |
| 44444444D | David | JEREZ | 0001 | ALMERÍA |
| 44444444D | David | JEREZ | 0002 | GRANADA |
| 44444444D | David | JEREZ | 0003 | CADIZ |

# Subconsultas

A veces tendrás que utilizar en una consulta los resultados de otra que llamaremos subconsulta.

SINTAXIS:

```
SELECT campos
    FROM tablas
    WHERE expresión OPERADOR
        ( SELECT listaExpr FROM tabla);
```

La subconsulta puede ir dentro de las cláusulas WHERE, HAVING o FROM.

# Subconsultas
## *De un valor*

La subconsulta puede devolver:

- **Un valor,** utiliza como **OPERADOR =,  >, <, >=, <=, !=, =**, sino dará error.

Ejemplo:

*-- Obtien los empleados cuyas pagas sean inferiores a lo que gana Maria.*

**SELECT** nombre_empleado**,** paga **FROM** EMPLEADOS

**WHERE** paga **< ( SELECT** paga **FROM** EMPLEADOS

**WHERE** nombre_empleado**=**'Maria'**);**

Lógicamente el resultado de la subconsulta debe incluir el campo que estamos analizando.

Se pueden realizar esas subconsultas las veces que haga falta, usando AND o OR.

# Subconsultas
## *De varios valores*

- **Varios valores,** más de un registro, se utiliza como **OPERADOR**:
  - **ANY**. Compara con cualquier registro de la subconsulta. La instrucción es válida si hay un registro en la subconsulta que permite que la comparación sea cierta.
  - **ALL**. Compara con todos los registros de la subconsulta. La instrucción resultará cierta si es cierta toda la comparación con los registros de la subconsulta.
  - **IN**. No utiliza comparador, lo que hace es comprobar si el valor se encuentra en el resultado de la subconsulta.
  - **NOT IN**. Comprueba si un valor no se encuentra en una subconsulta.

*Ejemplo: -- Obtiene el empleado que más cobra.*

```
SELECT nombre, sueldo FROM EMPLEADOS
WHERE sueldo >= ALL ( SELECT sueldo FROM EMPLEADOS );
```

# Combinaciones Especiales

Combina los resultados de 2 consultas en un solo resultado.

Si existen varios operadores se evalúan de izquierda a derecha.

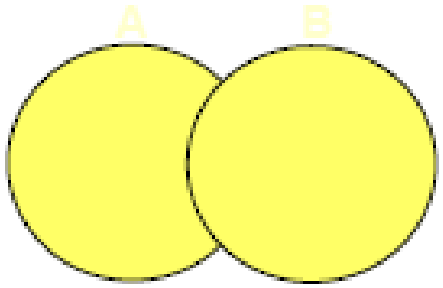
**UNION** → Combina los resultados de dos consultas eliminando las filas repetidas

**UNION ALL** → Devuelve todas las filas de ambas consultas incluyendo todas las entradas duplicadas

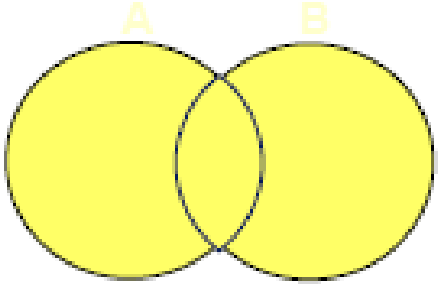
**INTERSECT** →Devuelve todas las filas comunes eliminando duplicados

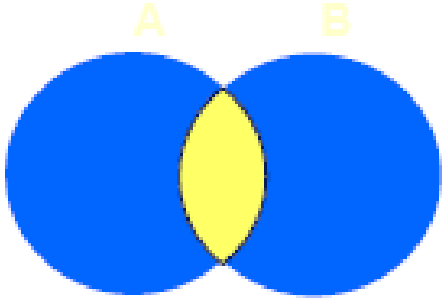
**MINUS** → Devuelve todas las filas del resultado de la primera consulta menos las contenidas en el resultado de la segunda consulta

# Combinaciones Especiales

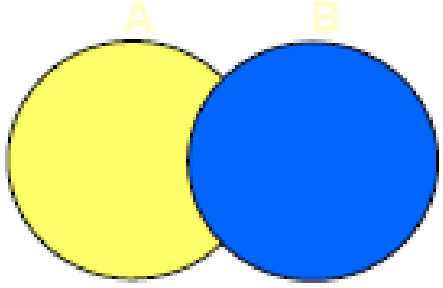



UNION/UNION ALL




INTERSECT




MINUS

# Combinaciones Especiales

Ejemplos:

```
-- Tipos y modelos que se encuentren el almacén 1, en el 2 o en ambos.
SELECT tipo,modelo FROM existencias WHERE n_almacen = 1
UNION
SELECT tipo,modelo FROM existencias WHERE n_almacen = 2;
-- Tipos y modelos que se encuentren en los almacenes 1 y 2 (en ambos).
SELECT tipo,modelo FROM existencias WHERE n_almacen = 1
INTERSECT
SELECT tipo,modelo FROM existencias WHERE n_almacen = 2;
-- Tipos y modelos que se encuentren el almacén 1 y no en el 2.
SELECT tipo,modelo FROM existencias WHERE n_almacen = 1
MINUS
SELECT tipo,modelo FROM existencias WHERE n_almacen = 2;
```

# Conversiones implícitas

Se puede convertir datos automáticamente a fin de que la expresión final tenga sentido. En ese sentido son fáciles las conversiones de texto a número y viceversa.

Ejemplos:

- SELECT **5+'3'** FROM DUAL → *El resultado es 8*;
- SELECT **concat (5,'3')** FROM DUAL; → *El resultado es 53*

Pero en determinadas ocasiones querremos realizar conversiones explícitas. Para hacerlo utilizaremos las funciones que se detallan a continuación.