UNIDAD 6 CONSULTAS BÁSICAS

1 EL LENGUAJE DE MANIPULACIÓN DE DATOS

En la Unidad 5 vimos una introducción al lenguaje SQL y los diferentes subconjuntos de instrucciones. Aprendimos los fundamentos del diseño físico de una base de datos relacional. En esta unidad 6 vamos a aprender los conceptos fundamentales de las consultas simples en una base de datos con la sentencia SELECT.

DML o Data Manipulation Language es una parte del lenguaje SQL que se encarga de la gestión de los datos almacenados.

Las sentencias DML del lenguaje SQL son las siguientes:

- La sentencia SELECT, que se usa para extraer información de la base de datos.
- La sentencia INSERT que inserta uno o varios registros en una tabla.
- La sentencia DELETE, que borra registros de una tabla.
- La sentencia UPDATE, que modifica registros de una tabla

2 LA SENTENCIA SELECT

La sentencia SELECT es una de las más versátiles del lenguaje SQL. Pertenece al grupo DML del lenguaje SQL y permite realizar consultas sobre una o varias tablas de las bases de datos.

Realizar una consulta en SQL, consiste en recuperar u obtener aquellos datos, que, almacenados en filas y columnas de una o varias tablas de una base de datos, cumplen unas determinadas condiciones. Esta sentencia permite:

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

2.1 Sintaxis básica de la sentencia SELECT

Para realizar cualquier consulta se utiliza la sentencia `SELECT`.

Formato inicial de la sentencia SELECT

```
SELECT [DISTINCT | ALL] * lista de elementos
FROM lista de tablas
[WHERE condición de selección]
[ORDER BY especificación para ordenar [ASC | DESC]
```

La consulta más sencilla consiste en recuperar una o varias columnas de una tabla.

Formato de una consulta sencilla

```
SELECT [ALL |DISTINCT] * lista de elementos
FROM tabla
```

donde:

- La cláusula **SELECT** permite elegir columnas y/o valores (resultados de expresiones)
- *lista de elementos* son nombres de columnas o expresiones obtenidas a partir de ellas, y separadas por comas.
- *, selecciona todas las columnas de la tabla.
- **ALL** obtiene los valores de todos los elementos seleccionados en todas las filas, aunque sean repetidos. Es la opción por defecto.

- **DISTINCT** obtiene los valores no repetidos de todos los elementos.
- **FROM** *tabla*, indica el nombre de la tablas en la que se realiza la consulta. Si la tabla no es de la propiedad del usuario, aunque tenga permiso de acceso, deberá usarse con nombre_propietario.tabla.. nombre_esquema.tabla

Veamos algunos ejemplos:

- `SELECT * FROM alumnos; – Selecciona todos los campos de la tabla alumnos`
- `SELECT nombre, apellidos, 1+2 FROM alumnos;`

  `– Realizamos operaciones`
- `SELECT 1+6;` (en MySQL) `SELECT 1+6 FROM dual;` (en Oracle)

ALIAS DE TABLA

SQL permite asignar más de un nombre a la misma tabla, dentro de la misma consulta. Usar alias para una tabla es opcional cuando su finalidad consiste en simplificar el nombre original y obligatorio en consultas cuya sintaxis lo requiera.

```
SELECT ... FROM Empleados e
```

ALIAS DE COLUMNA

Los títulos o cabeceras que muestra la salida de una consulta para las columnas seleccionada, se corresponden con los nombres de las columnas de las tablas. Para mejorar su legibilidad y estética se utilizan los alias de columnas. El alias se escribe detrás de la columna, puede ir antepuesto por la palabra AS. Si el alias comprenden más de una palabra deberá ir dentro de dobles comillas.

```
SELECT apellido, salario + comisión AS Importe_Total FROM Empleados.
```

En el ejemplo anterior se mostraría el salario + comisión como Importe_Total

3 CONSULTAS CALCULADAS

3.1 Cálculos aritméticos

Los operadores + (suma), - (resta), * (multiplicación), / (división) y % (resto de división entera), se pueden utilizar para hacer cálculos en las consultas. Cuando se utilizan como expresión en una consulta SELECT, no modifican los datos originales sino que como resultado de la vista generada por SELECT, parece una nueva columna. Ejemplo:

```
SELECT apellido, salario*1.25 AS Salario_incremento FROM Empleados.
```

La prioridad de estos operadores es la normal: tiene más prioridad la multiplicación y división, después la suma y la resta. En caso de igualdad de prioridad, se realiza primero la operación que está más a la izquierda. Como es lógico se puede evitar cumplir esta prioridad usando paréntesis, el interior de los paréntesis es lo que se ejecuta primero.

Cuando una expresión aritmética se calcula sobre valores NULL, el resultado es el propio valor NULL.

3.2 Concatenación de textos

Todos los SGBD incluyen algún operador para concatenar textos. En SQLServer es el sigo +, el Oracle son los signos ||. En algunas versiones de MySQL funciona el símbolo ||, en otras versiones hay que utilizar una función, la función CONCAT, que también funciona en casi todos los SGBD.

4 CONDICIONES

Para seleccionar las filas de la tabla sobre las que se va a realizar una consulta, la cláusula `WHERE` permite incorporar una condición de selección a la sentencia `SELECT`. Esta sentencia permite colocar una condición que han de cumplir todos los registros, los que no la cumplan no aparecen en el resultado.

Formato de consulta con condición de selección

```
SELECT [ALL | DISTINCT] lista de elementos
FROM lista de tablas
WHERE condición de selección.
```

donde:

- ***condición de selección*** es una expresión formada por columnas de la tabla, constantes, funciones, operadores aritméticos, operadores de comparación y operadores lógicos, que deberá ser cierta para que una fila de la tabla sea seleccionada como parte de la salida obtenida por la consulta.

4.1 Operadores de comparación

Se pueden utilizar en la sentencia o clausula `WHERE`, son:

Operador	Significado
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
=	Igual
<>	Distinto
!=	Distinto

Se pueden utilizar tanto para comparar números como para comparar textos y fechas. En el caso de los textos, las comparaciones se hacen en orden alfabético. Solo que es un orden alfabético estricto. Es decir, el orden de los caracteres en la tabla de códigos.

En muchos SGBD hay problemas con la ñ y otros símbolos nacionales (en especial al ordenar oi comparar con el signo de mayor o menor, ya que el orden ASCII no respeta el orden de cada alfabeto nacional). No obstante es un problema que tiende a arreglarse en la actualidad en todos los SGBD, especialmente si son compatibles con Unicode.

4.2 Operadores lógicos

Son:

Operador	Significado
AND	Devuelve verdadero si las expresiones a su izquierda y derecha son ambas verdaderas
OR	Devuelve verdadero si cualquiera de las dos expresiones a izquierda y derecha son ambas verdaderas
NOT	Invierte la lógica de la expresión que está a su derecha. Si era verdadera, mediante NOT pasa a ser falso.

Ejemplos:

```
/* Obtener a las personas de entre 25 y 50 años*/
SELECT nombre, apellidos FROM personas WHERE edad>=25 AND edad<=50;
/* Obtener a las personas de más de 60 años o de menos de 20 años*/
```

```
SELECT nombre, apellidos FROM personas WHERE edad>60 OR edad<20;
```

4.3 Operador LIKE

Se utiliza para compara una expresión de cadena con un modelo en una expresión SQL.

Su sintaxis es:

```
Expresión LIKE modelo
```

El operador usa diferentes máscaras:

Carácter	Descripción	Ejemplo
Comodín %	Cualquier cadena de cero o más caracteres	`WHERE título LIKE '%equipo%'`, busca todos los títulos que contengan la palabra equipo en cualquier parte del título.
Subrayado _	Cualquier carácter individual	`WHERE nombre LIKE '_ean'`, busca todos los nombres de 4 letras que finalicen con ean (Dean, Sean, etc.)

4.4 Operador IN

Devuelve aquellos registros cuyo campo indicado coincide con alguno de los indicados en una lista. Su sintaxis es:

```
Campo [NOT] IN (valor1, valor2,..)
```

Ejemplo

```
/* Obtener las piezas cuyos precios sean 3, 6 u 8 (no valen el precio 4 ni el 6)*/
SELECT * FROM piezas WHERE pecio_venta IN (3,5,8);
```

4.5 Operador BETWEEN

Devuelve aquellos registros según el intervalo de valores de un campo. Su sintaxis es:

```
Campo [NOT] BETWEEN valor1 AND valor2
```

Ejemplo

```
/* Obtener las piezas cuyos precios estén entre 3 y 8 (ambos incluidos)*/
SELECT * FROM piezas WHERE pecio_venta BETWEEN 3 AND 8;
```

4.6 Operador NULL y NOT NULL

Se dice que una columna de una fila es NULL si está completamente vacía. Para comprobar si el valor de una columna es nulo empleamos la expresión:

```
columna IS NULL
```

Si queremos saber si el valor de una columna no es nulo, usaremos la expresión:

```
columna IS NOT NULL
```

Cuando comparamos con valores nulos o no nulos no podemos utilizar los operadores de igualdad, mayor o menor.

4.7 Precedencia de operadores

A veces las expresiones que se producen en los SELECT son muy extensas y es difícil saber qué parte de la expresión

se evalúa primero, por ellos se indica la siguiente tabla de precedencias (tomada de Oracle):

Orden de precedencia	Operador
1	* (Multiplicar) / (dividir)
2	+ (suma) - (resta)
3	|| Concatenación
4	Comparaciones (>, <, !=, ...)
5	IS [NOT] NULL, [NOT] LIKE, IN
6	NOT
7	AND
8	OR

4.8 Condición de selección por límite de número de registros

Este tipo de filtro no es estándar y depende del SGBD con el que estemos trabajando. Consiste en limitar el número de registros devuelto por una columna.

En MySQL es:

```
[LIMIT [desplazamiento, ] nfilas]
```

donde:

- *nfilas* especifica el número de filas a devolver
- *desplazamiento* especifica a partir de que fila se empieza a contar.

En Oracle, se limita el número de filas apoyándose en una pseudocolumna, de nombre rownum.

Ejemplo:

```
/*Sacar los 5 primeros empleados*/
SELECT * FROM empleados WHERE rownum<=5;
```

5 ORDENACIÓN

El orden inicial de los registros obtenidos por cuna consulta con SELECT no guarda más que una relación respecto al orden en el que fueron introducidos. Para obtener la salida de una consulta clasificada por algún criterio o especificación, la sentencia SELECT dispone de la cláusula **ORDER BY** para ordenar.

Formato de consulta con ordenación

```
SELECT [ALL | DISTINCT] * lista de elementos
FROM lista de tablas
WHERE condición de selección
ORDER BY especificación para ordenar [ASC | DESC]
```

donde:

- *especificación para ordenar* es una lista de columnas o expresiones obtenidas a partir de ellas, separadas por comas, y cada una de ellas con indicación del tipo de ordenación.
- tipo de ordenación: **ASC** (ascendente) o **DESC** (descendente). Por omisión es ASC. Si las especificación para ordenar contiene más de una columna o expresión, el orden en que se realizan la clasificaciones es de izquierda a derecha.

6 NULOS

En los lenguajes de programación se utiliza el valor nulo para reflejar que un identificador (una variable, un objeto,..) no tiene ningún contenido. Por ejemplo cuando un puntero en lenguaje C señala a 'null' se dice que no está señalando a nada. Al programar en esos lenguajes no se permite utilizar un valor nulo en operaciones aritméticas o lógicas.

Las bases de datos relacionales permiten más posibilidades para el valor nulo (null), aunque su significado no cambia: valor vacío. No obstante en las bases de datos se utiliza para diversos fines.
En claves ajenas o secundarias indican que el registro actual no está relacionado con ninguno. En otros atributos indica que la tupla en cuestión carece de dicho atributo: por ejemplo en una tabla de personas un valor nulo en el atributo teléfono indicaría que dicha persona no tiene teléfono.
Es importante indicar que el texto vacío '', no significa lo mismo en un texto que el nulo; como tampoco el valor cero significa nulo.
Puesto que ese valor se utiliza continuamente, resulta imprescindible saber cómo actúa cuando se emplean operaciones lógicas sobre ese valor. Eso significa definir un tercer valor en la lógica booleana, además de los clásicos verdadero y falso. Un valor nulo no es ni verdadero ni falso (se suele interpretar como un quizás, o usando la aritmética clásica en valores lógicos, el 1 es verdadero, el 0 falso y el 0,5 nulo).
El uso de operadores lógicos con el nulo da lugar a que:

- **verdadero Y (AND) nulo** da como resultado, nulo (siguiendo la aritmética planteada antes: $1 \cdot 0{,}5 = 0{,}5$)
- **falso Y (AND) nulo** da como resultado, falso ($0 \cdot 0{,}5 = 0$)
- **verdadero O (OR)** nulo da como resultado, verdadero ($1 + 0{,}5 > 1$)
- **falso O nulo** da como resultado nulo ($0 + 0{,}5 = 0{,}5$)
- **la negación de nulo**, da como resultado nulo

7 SUBCONSULTAS

7.1 ¿Qué es una subconsulta?

Una subconsulta en SQL consiste en utilizar los resultados de una consulta dentro de otra, que se considera la principal. Las subconsultas se utilizan para:

- Dividir una consulta compleja en varios pasos lógicos.
- Responder una consulta que depende de los resultados de otra consulta.

7.2 ¿Cómo utilizar las subconsultas?

Cuando se utilizan subconsultas, hay que tener en cuenta los siguientes hechos e instrucciones:

- Las subconsultas se deben incluir entre paréntesis.
- La subconsulta se ejecuta primero y, posteriormente el valor extraído es utilizado en la consulta principal.
- Se puede utilizar subconsultas en lugar de una expresión siempre y cuando se devuelva un solo valor o una lista de valores.
- Pueden tenerse subconsultas dentro de subconsultas, con una anidación de hasta 32 niveles, el límite varía dependiendo de la memoria disponible.

7.3 Subconsultas simples

Formato de una subconsulta simple

```
SELECT listaExpresiones
FROM tabla
WHERE expresion OPERADOR
(SELECT [ALL | DISTINCT] listaExpresiones
FROM TABLA
WHERE condición de selección)
```

donde el formato de la sentencia SELECT entre paréntesis tiene las siguientes diferencias con la sentencia SELECT de las consultas:

- No tiene sentido la cláusula ORDER BY ya que los resultados de una subconsulta se utilizan internamente y no son visibles al usuario.
- Los nombres de columna que aparecen en una subconsulta pueden referirse a columnas de la tabla de la consulta principal y se conoce como *referencia externa.*

Se puede colocar el SELECT de la subconsulta dentro de las cláusulas WHERE, FROM o HAVING (se verá en la Unidad 7). El operador puede ser >, >=, <, <=, !=, = o IN.

Ejemplo:

```
SELECT nombre, sueldo FROM empleados
WHERE sueldo < (SELECT sueldo FROM empleados WHERE nombre='Silvia');
```

Lógicamente el resultado de la subconsulta debe incluir el campo que estamos analizando. Se pueden realizar esas subconsultas las veces que haga falta:

```
SELECT nombre, sueldo FROM empleados
WHERE sueldo < (SELECT sueldo FROM empleados WHERE nombre='Silvia')
AND sueldo > (SELECT sueldo FROM empleados WHERE nombre='Ana')
```

Las subconsultas siempre se deben encerrar entre paréntesis y se debería colocar a la derecha del operador relacional.

Condición de selección con operadores aritméticos de comparación

Se utiliza cuando la subconsulta devuelve un único valor a comparar con una expresión, por lo general formada a partir de una fila obtenida en la consulta principal. Si la comparación resulta cierta (TRUE), la condición de selección también lo es. Si la subconsulta no devuelve ninguna fila (NULL), la comparación devuelve también el valor NULL.

Formato para la condición de selección de operadores aritméticos de comparación

```
expresión operador aritmético de comparación subconsulta
operadores aritméticos de comparación = < > <> >= <=
```

Estos operadores comparan el valor de una expresión con un valor único producido por una subconsulta.

7.4 Valores de retorno de las subconsultas y condiciones de selección

Una subconsulta puede formar parte de la condición de selección en las cláusulas WHERE o HAVING.

El resultado de una subconsulta puede ser un valor simple o más de un valor. Según el retorno de la subconsulta, el operador de comparación que se utiliza en la condición de selección del WHERE o HAVING deberá ser del tipo apropiado según la tabla siguiente:

Operador Comparativo	Retorno de la subconsulta
De tipo aritmético	Valor simple
De tipo lógico	Más de un valor

7.5 Subconsultas de múltiples filas

En ocasiones se necesitan subconsultas que devuelvan más de una fila. Por ejemplo, mostrar el sueldo y el nombre de los empleados cuyo sueldo supera el de cualquier empleado del departamento de ventas.

La subcosulta necesaria mostraría todos los sueldos del departamento de ventas. Aquí no podemos utilizar un operado de comparación, hay que utilizar otro tipo de operadores, unos operadores lógicos especiales, que permiten el uso de subconsultas de varias filas. Veamos estos operadores:

OPERADOR LÓGICO IN

Comprueba si los valores de la fila actual de la consulta principal coinciden con algunos de los valores devueltos por la subconsulta. Si el resultado es afirmativo, la comparación resulta cierta (TRUE)

Formato para la condición de selección con el operador lógico IN

```
expresión [NOT] IN subconsulta
```

Ejemplos:

```
/* Obtener los nombres de los empleados cuyos dni estén en la tabla de directivos*/
SELECT nombre, sueldo FROM empleados
WHERE dni IN (SELECT dni FROM directivos);
```

Si se necesita comprobar dos columnas en una consulta IN, se hace:

```
SELECT nombre, sueldo FROM empleados
WHERE (cod1, cod2) IN (SELECT cod1, cod2 FROM directivos);
```

OPERADORES LÓGICOS ANY Y ALL

Se utilizan junto con los operadores aritméticos de comparación para ampliar las posibles comprobaciones de valores obtenidos a partir de la fila seleccionad en la consulta principal con valores obtenidos en la subconsulta. Su uso es a menudo sustituido por el operador IN.

Formato para la condición de selección con el operador lógico ANY / ALL

```
expresión operador aritmético de comparación {ANY |ALL} subconsulta
operadores aritméticos de comparación = < > <> >= <=
```

El **operador ANY** con uno de los seis operadores aritméticos compara el valor de la expresión formada a partir de la consulta principal con valores producidos por la subconsulta. Si alguna de las comparaciones individuales produce un resultado verdadero (TRUE), el operador ANY devuelve un resultado verdadero (TRUE)

El **operador ALL** también se utiliza con los operadores aritméticos para comprobar un valor de la expresión formada a partir de la consulta principal con cada uno de los valores de datos producidos por la subconsulta. Si todos los resultados de las comparaciones son ciertos (TRUE), ALL devuelve en valor cierto (TRUE).

Ejemplo:

```
/* Visualizar el nombre y el sueldo del empleaod que más cobra*/

SELECT nombre, sueldo FROM empleados

WHERE sueldo >=ALL (SELECT sueldo FROM empleados);
```

7.6 Operador Lógico EXISTS

Se utiliza cuando la condición de selección consiste exclusivamente en comprobar que la subconsulta devuelve alguna fila seleccionada según la condición de la propia subconsulta. El operador EXISTS no necesita que la subconsulta devuelva alguna columna porque no utiliza ninguna expresión de comparación, justificando así que casi siempre aparezca * en el formato de la misma

Formato para la condición de selección con el operador lógico EXISTS

```
expresión [NOT] EXISTS subconsulta
```

Una subconsulta expresada con el operador EXISTS también podría expresarse con el operador IN

Ejemplo:

```
/* Visualizar las piezas que se encuentran en la tabla existencias */

SELECT tipo, modelo, precio_venta FROM piezas p

WHERE EXISTS (SELECT tipo, modelo FROM existencias

            WHERE tipo=p.tipo AND modelo=p.modelo);
```

La tabla piezas necesita llevar un alias para poder evaluar la condición de selección de la subconsulta ya que tanto en piezas como en existencias hay columnas con el mismo nombre (tipo y modelo)

```
/* Visualizar las piezas que no se encuentran en la tabla existencias */

SELECT tipo, modelo, precio_venta FROM piezas p

WHERE NOT EXISTS (SELECT tipo, modelo FROM existencias

            WHERE tipo=p.tipo AND modelo=p.modelo);
```

7.7 Subcosultas correlacionadas

Cuando los nombres de columnas que aparecen en una subconsulta son nombres de columnas de la consulta principal o de otra subconsulta más externa, caso de las anidadas, se dice que son *referencias externas* y la subconsulta que es *correlacionada*.

En las subconsultas correlacionadas, cada vez que se selecciona una nueva fila, en la consulta principal o en otra subconsulta más externa que contenga la columna referenciada, se repetirá el proceso de selección.

Si en una subconsulta correlacionada coincide el nombre de una referencia externa con el nombre de alguna columna de la tabla que está siendo seleccionada en la subconsulta se deberá asignar un alias a cada tabla y se utilizará para identificar cada columna

7.8 Subconsultas anidadas

Cuando una subconsulta forma parte de una condición de selección en una cláusula WHERE o HAVING de otra subconsulta. Se dice que **es una subconsulta anidada.**

8 CONSULTAS MULTITABLA

Hasta ahora, en las consultas que hemos realizado sólo se ha utilizado una tabla, indicada a la derecha de la palabra de la palabra FROM; pero hay veces que una consulta necesita columnas de varias tablas. En este caso se expresarán a la derecha de la palabra FROM.

8.1 Producto cartesiano de tablas

Para poder acceder a dos o más tablas de una base de datos, SQL genera internamente una tabla en la que cada fila de una tabla se combina con todas y cada una de las filas de las demás tablas. Esta operación se conoce como **producto cartesiano** y la tabla resultante contiene todas las columnas de todas las tablas que se han multiplicado.

La multiplicación de tablas es también conocida como **composición o combinación de tablas.**

Produce una nueva tabla con todas las columnas de las tablas implicadas en el FROM, En la nueva tablas aparecerán todos los registros de cada tabla relacionados con todos los registros de las otra tablas.

Ejemplo: Obtener todos los empleados con su nombre de departamento y su localidad

```
SELECT emp_no AS "N° empleado", apellido, dnombre AS Departamento,
localidad FROM empleados, departamentos;
```

El producto cartesiano a veces es útil para realizar consultas complejas, pero en el caso normal no lo es, se hace necesario discriminar es producto para que solo aparezcan los registros de los empleados con sus departamentos correspondientes. A esto se le llama asociar (join) tablas.

8.2 Asociar tablas

Asociar tablas es usar un producto cartesiano para obtener una información a la que se le impone una condición. Esta condición de asociación se escribe en la cláusula **WHERE**.

Ejemplo 1:

```
SELECT emp_no AS "N° empleado",apellido, dnombre AS Departamento,      localidad  FROM
empleados e, departamentos d WHERE e.dep_no = d.dep_no
```

SQL resuelve el anterior ejercicio con el siguiente proceso:

La cláusula FROM genera todas las combinaciones posibles de filas de la tabla de *empleados* (9 filas) por las de la tabla de *departamentos* (4 filas), resultando una tabla producto de 4x9=36 filas.

La cláusula WHERE (o la ON) selecciona únicamente aquellas filas de la tabla producto donde coinciden los números de departamento, que necesitan del alias por tener el mismo nombre en ambas tablas. En total se han seleccionado 9 filas y las 27 restantes se eliminan.

La sentencia SELECT visualiza las columnas especificadas de la tablas producto para las filas seleccionadas.

Ejemplo 2: Supongamos las siguiente tablas:

empleados

dni_ empleado	nombre_empleado	sueldo

tareas

cod_tarea	descripcion_tarea	dni_empleado

utensilios_utilizados

cod_tarea	nombre_utensilio

Una asociación de estas tablas, podría ser la siguiente consulta:

```
SELECT cod_tarea, descripcion_tarea, dni_empleado,
nombre_empleado
FROM tareas,empleados
WHERE tareas.dni_empleado = empleados.dni;
```

Se utiliza la notación **tabla.columna** para evitar la ambigüedad, ya que el mismo nombre de campo se puede repetir en varias tablas. Para evitar repetir continuamente el nombre de la tabla, se puede utilizar un alias de tabla:

```
SELECT t.cod_tarea, t.descripcion_tarea, e.dni, e.nombre_empleado
FROM tareas t,empleados e
WHERE t.dni_empleado = e.dni;
```

Al apartado **WHERE** se le pueden añadir condiciones encadenándolas con el operador AND.
Ejemplo:

```
SELECT t.cod_tarea, t.descripcion_tarea
FROM tareas t,empleados e
WHERE t.dni_empleado = e.dni AND  e.nombre_empleado='Javier';
```

Es posible enlazar más de dos tablas a través de sus campos relacionados.
Ejemplo:

```
SELECT t.cod_tarea, t.descripcion_tarea, e.nombre_empleado, u.nombre_utensilio
FROM tareas t,empleados e, utensilios_utilizados u
WHERE t.dni_empleado = e.dni AND t.cod_tarea=u.cod_tarea;
```

8.3 Relaciones sin igualdad

La condición de selección que establezcamos para componer o combinar tablas no tiene por qué ser siempre mediante el operador aritmético de igualdad, aunque su uso sea el más frecuente. SQL permite utilizar cualquier operador aritmético de comparación.

Ejemplo: Listar los empleados de departamentos con códigos mayores que el código del departamento de contabilidad.

```
SELECT e.emp_no "N°Empleado",e.apellido
FROM empleados e,departamentos d
```

```
WHERE d.dnombre='CONTABILIDAD' AND e.dep_no>d.dep_no
```

8.4 Notación SQL2

En la versión SQL de 1999 se ideó una nueva sintaxis para consultar varias tablas. La razón fue separar las condiciones de asociación respecto de las condiciones de selección de registros. Oracle incorpora totalmente esta normativa.
La sintaxis completa es:

```
SELECT tabla1.columna1, tabl1.columna2,... tabla2.columna1, tabla2.columna2,...
FROM tabla1 [CROSS JOIN tabla2]| [NATURAL JOIN tabla2]|
    [JOIN tabla2 USING(columna)]|
    [JOIN tabla2 ON (tabla1.columa=tabla2.columna)]|
    [LEFT|RIGHT|FULL OUTER JOIN tabla2 ON (tabla1.columa=tabla2.columna)]
```

8.4.1 Cross Join

Utilizando la opción CROSS JOIN se realiza un producto cruzado entre las tablas indicadas. Eso significa que cada tupla de la primera tabla se combina con cada tupla de la segunda tabla. Es decir si la primera tabla tiene 10 filas y la segunda otras 10, como resultado se obtienen 100 filas, resultado de combinar todas entre sí.

```
SELECT emp_no AS "N° empleado", apellido, dnombre AS Departamento,     localidad
FROM empleados CROSS JOIN departamentos
```

8.4.2 Inner Join o Composición interna

El ejemplo anterior no tiene ni sentido ni aplicación, la solución correcta del ejemplo sería:

```
SELECT emp_no AS "N° empleado",apellido, dnombre AS Departamento,      localidad
FROM empleados e INNER JOIN departamentos d ON e.dep_no =  d.dep_no
```

Una composición o combinación (**join**) **consiste en aplicar una condición de selección a las filas** obtenidas de la multiplicación de las tablas sobre las que se está realizando una consulta.

La composición más sencilla es aquella en que la condición de selección se establece con el operador de igualdad entre las columnas que deban coincidir exactamente en tablas diferentes. Esta composición es una ***equicomposición*** (**EQUAL JOIN**).

La composición más sencilla es aquella en que la condición de selección se establece con el operador de igualdad entre las columnas que deban coincidir exactamente en tablas diferentes. Esta composición es una ***equicomposición*** (**EQUAL JOIN**).

Formato de una equicomposición

```
SELECT columnas seleccionadas de las tablas

FROM tabla1 alias1,tabla2 alias2,...

WHERE { tabla1.columna=tabla2.columna|alias1.columna=alias2.columna};
```

Reglas de composición

- Pueden unirse tantas tablas como se desee (aunque la experiencia aconseja que no sean más de 8 por optimización).

- El criterio de emparejamiento para las tablas se denomina *predicado o criterio de composición.*
- Puede usarse más de una pareja de columnas para especificar la composición entre dos tablas cualesquiera.
- SQL no exige que las columnas de emparejamiento estén relacionadas como clave primaria y clave ajena, aunque es lo más habitual. Pueden servir cualquier par de columnas de dos tablas, siempre que tengan tipos de datos comparables.
- En la SELECT pueden seleccionarse columnas de ambas tablas.
- Si hay columnas con el mismo nombre en las distintas tablas de la FROM, deben identificarse como *nombre de tabla.nombre de columna o* utilizar una alias para cada tabla.

Estas reglas se aplican a todo tipo de composición o combinación.

En una composición INNER JOIN podemos usar:

- USING Permite establecer relaciones indicando qué columna (o columnas) común a las dos tablas hay que utilizar. Las columnas deben tener exactamente el mismo nombre en ambas tablas.

```
SELECT * FROM piezas JOIN existencias USING (tipo, modelo)
```

- ON: Permite establecer relaciones cuya condición se establece manualmente, lo que permite realizar asociaciones más complejas o bien asociaciones cuyos campos en las tablas no tienen el mismo nombre.

```
SELECT * FROM piezas JOIN existencias

ON (piezas.tipo=existencias.tipo AND piezas.modelo=existencias.modelo)
```

8.4.3 Natural Join o Composiciones naturales

Es una especialización de la anterior. En este caso se comparan todas las columnas que tengan el mismo nombre en ambas tablas. La tabla resultante contiene sólo una columna por cada par de columnas con el mismo nombre.

En este caso se comparan todas las columnas que tengan el mismo nombre en ambas tablas. La tabla resultante contiene sólo una columna por cada par de columnas con el mismo nombre.

```
SELECT emp_no AS "N° empleado", apellido, dnombre AS Departamento,     localidad
FROM empleados NATURAL JOIN departamentos
```

8.4.4 Composición externa

Hasta ahora el concepto que conocemos de JOIN corresponde a lo que se conoce como JOIN INTERNO (INNER JOIN). Pero cuando se realiza una composición o combinación de tablas estableciendo una determinada relación entre sus columnas, puede ocurrir que no se emparejen todas las filas que debieran por faltar algunas de ellas. Es aconsejable que la salida, obtenida por una consulta en la que se pudiera presentar esta posibilidad, muestre todas las filas, aunque algunas con falta de información. Para conseguir este resultado se utiliza la composición o **combinación externa (OUTER JOIN)**.

Si las filas (o registros) que admiten no tener correspondencia son los que aparecen en la tabla de la izquierda se

llama composición de tabla izquierda o **LEFT JOIN** (o **LEFT OUTER JOIN**)

Se produce entre dos tablas ofreciendo la siguiente información:

- El JOIN INTERNO (o JOIN) de las dos tablas
- Por cada fila de la primera tabla que no haya correspondido a ninguna fila de la segunda tabla, se añade una fila a los resultados, utilizando los valores de la columna de la primera tabla y suponiendo un valor NULL para todas las columnas de la segunda tabla correspondientes en el emparejamiento.

Sintaxis

```
FROM tabla1 LEFT JOIN tabla2 ON tabla1.campo1 operadorComparación tabla2.campo2
```

Ejemplo: Queremos conocer todos los datos de los clientes y sus pedidos, pero que aparezcan los clientes que no han realizado pedidos

```
SELECT c.*, p.* FROM clientes c LEFT JOIN pedidos p
    on c.cliente_no=p.cliente_no
```

Si las filas (o registros) que admiten no tener correspondencia son los que aparecen en la tabla de la derecha se llama composición de tabla derecha o **RIGHT JOIN** (o **RIGHT OUTER JOIN**)

Se produce entre dos tablas ofreciendo la siguiente información:

- El JOIN INTERNO (o JOIN) de las dos tabla.
- Por cada fila de la segunda tabla que no haya correspondido a ninguna fila de la primera tabla, se añade una fila a los resultados, utilizando los valores de la columna de la segunda tabla y suponiendo un valor NULL para todas las columnas de la primera tabla correspondientes en el emparejamiento.

Sintaxis

```
FROM tabla1 RIGHT JOIN tabla2 ON tabla1.campo1 operadorComparación tabla2.campo2
```

Ejemplo: Queremos ver todos los departamentos con los nombres de los directores:

```
SELECT apellido, dnombre FROM EMPLEADOS e RIGHT JOIN DEPARTAMENTOS d
on e.dep_no=d.dep_no;
```

La representación gráfica de una composición o combinación externa depende del SQL utilizado.

En ORACLE y en SQLBASE se coloca (+) detrás de la columna en cuya tabla puede faltar información:

```
WHERE tabla1.columna1(+)=tabla2.columna2 o
WHERE tabla1.columna1=tabla2.columna2(+)
```

JOIN EXTERNO COMPLETO

Un JOIN EXTERNO COMPLETO es una ampliación de un JOIN INTERNO que contiene la siguiente información:

El JOIN INTERNO (o JOIN) de las dos tablas

Por cada fila de la primera tabla que no haya correspondido a ninguna fila de la segunda tabla, se añade una fila a los resultados, utilizando los valores de la columna de la primera tabla y suponiendo un valor NULL para todas las columnas de la segunda tabla correspondientes en el emparejamiento.

Por cada fila de la segunda tabla que no haya correspondido a ninguna fila de la primera tabla, se añade una fila a los resultados, utilizando los valores de la columna de la segunda tabla y suponiendo un valor NULL para todas las columnas de la primera tabla correspondientes en el emparejamiento.

Sintaxis

```
FROM tabla1 FULL OUTER JOIN tabla2 ON tabla1.campo1 operadorComparación tabla2.campo2
```

8.5 Composiciones o combinaciones de una tabla consigo misma

Si desde una fila de una tabla podemos acceder a otra fila de la misma tabla estamos realizando una composición o combinación de una tabla consigo misma.

Ejemplo. Obtener la lista de los empleados con los nombres de sus directores.

```
SELECT e1.emp_no AS "N°Empleado",e1.apellido, e1.director AS "N°Director", e2.apellido AS "Nombre Director"
FROM empleados e1,empleados e2
WHERE e1.director=e2.emp_no;
```

Cada empleado de la tabla tiene una columna para su número de empleado (*emp_no*) y otra para el número de empleado de su director (*director*).

A partir del dato de la columna *director* de un empleado se puede acceder a otro empleado que contenga el mismo dato en su columna *emp_no*. Las dos filas de la tabla se están relacionando a través de las columnas *director* y *emp_no.*

El uso de alias es obligado por tratarse de la misma tabla donde todas las filas tienen las mismas columnas.

Usando la palabra clave INNER JOIN, esta consulta queda:

```
SELECT e1.emp_no AS "N°Empleado",e1.apellido, e1.director AS "N°Director", e2.apellido AS "Nombre Director"
FROM empleados e1 INNER JOIN empleados e2
ON e1.director=e2.emp_no;
```

8.6 Composiciones y subconsultas

Hay ocasiones en que una consulta puede resolverse con una composición o combinación (join) de tablas, o con una subconsulta. En este caso sería preferible hacerlo con una subconsulta.

En general, si no se necesita visualizar columnas de más de una tabla, se utiliza una subconsulta.

Si se necesita visualizar columnas de más de una tabla, se usará una composición o combinación.

Ejemplo.

Obtener apellido y oficio de los empleados que tienen el mismo oficio y mismo número de departamento que el de INVESTIGACIÓN.

Con subconsulta:

```
SELECT apellido,oficio FROM empleados
WHERE oficio IN (SELECT oficio FROM empleados WHERE dep_no IN
                        (SELECT dep_no FROM departamentos
                                WHERE dnombre='INVESTIGACION'))
```

Con composición de tablas:

```
SELECT apellido,oficio
FROM empleados
WHERE oficio IN (SELECT e.oficio
                  FROM empleados e,departamentos d
                  WHERE e.dep_no=d.dep_no AND
                  d.dnombre='INVESTIGACION');
```