Preparador Informática

www.preparadorinformatica.com

MANUAL 4

BASES DE DATOS

SQL (Avanzado 1ª parte)

1. Consultas de datos. SELECT

Para recuperar información o, lo que es lo mismo, para realizar consultas a la base de datos, utilizaremos una única sentencia SELECT. El usuario emplea esta sentencia con el nivel de complejidad apropiado para él: especifica qué es lo que quiere obtener, no dónde ni cómo. De la consulta se puede obtener: cualquier unidad de datos, todos los datos, cualquier subconjunto de datos, cualquier conjunto de subconjuntos de datos.

El formato de la sentencia SELECT es el siguiente:

```
SELECT [ALL|DISTINCT]
[expre_colum1, expre_colum2, ..., expre_column | * ]
FROM [nombre_tabla1, nombre_tabla2, ..., nombre_tablan]
[WHERE condición]
[ORDER BY expre_colum [DESC|ASC] [,expre_colum [DESC|ASC]]...];
```

Donde expre_colum puede ser una columna de una tabla, una constante, una expresión aritmética, una función o varias funciones anidadas.

La única cláusula de la sentencia SELECT que es obligatoria es la cláusula FROM, el resto son opcionales.

- FROM: `FROM [nombre_tabla1, nombre_tabla2, ..., nombre_tablan]`

Especifica la tabla o lista de tablas de las que se recuperarán los datos. Por ejemplo, consultamos los nombres de alumnos y su nota en la tabla ALUMNOS:

```
SELECT NOM_ALUM, NOTA FROM ALUMNOS;
```

Es posible asociar un nuevo nombre a las tablas mediante alias. Por ejemplo, si la tabla ALUMNOS le damos el nombre de A, las columnas de la tabla irán acompañadas de A.

```
SELECT A.NOM_ALUM, A.NOTA FROM ALUMNOS A;
```


- WHERE: `[WHERE condición]`

Obtiene las filas que cumplen la condición expresada. La complejidad de la condición es prácticamente ilimitada. El formato de la condición es: *expresión operador expresión*. Las expresiones pueden ser una constante, una expresión aritmética, un valor nulo o un nombre de columna. Se pueden construir condiciones múltiples usando los operadores lógicos booleanos estándares: AND, OR y NOT. Recordemos los operadores, aunque ya se vieron en el manual 1. Está permitido emplear paréntesis para forzar el orden de evaluación.

Operadores de comparación.	
Operadores:	**Descripción:**
<	Menor que.
>	Mayor que.
< >	Distinto de.
< =	Menor o igual.
> =	Mayor o igual.
=	Igual.
BETWEEN	Se utiliza para especificar un intervalo de valores.
LIKE	Se utiliza para comparar.
IN	Se utiliza para especificar filas de una base de datos.

Operadores lógicos.	
Operadores:	**Descripción:**
AND	Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Devuelve el valor contrario de la expresión.

Veamos unos ejemplos de condiciones en la cláusula WHERE:

```
WHERE NOTA = 5
WHERE (NOTA>=10) AND (CURSO=1)
```

- ORDER BY:
  [ORDER BY expre_columna [DESC|ASC] [,expre columna [DESC|ASC]]...]

Esta cláusula especifica el criterio de clasificación del resultado de la consulta. ASC especifica una ordenación ascendente, y DESC descendente. Es posible anidar los criterios. El situado más a la izquierda será el principal. Por ejemplo:

```
SELECT * FROM ALUMNOS ORDER BY NOM_ALUM, CURSO DESC;
```

(ordena por NOM_ALUM ascendente y por CURSO descendente).

También se puede indicar mediante un número, que indica la posición de la columna a la derecha de SELECT, el criterio de clasificación. Por ejemplo:

```
SELECT DEPT_NO, DNOMBRE, LOC FROM DEPART ORDER BY 2;
```

(ordena la salida por la segunda columna que es DNOMBRE).

- ALL: Con la cláusula ALL recuperamos todas las filas, aunque algunas estén repetidas. Es la opción por omisión.

- DISTINCT: Sólo recupera las filas que son distintas.

```
SELECT DISTINCT DEPT_NO FROM EMPLE;
```

- Alías: Cuando se consulta la base de datos, los nombres de las columnas se usan como cabeceras de presentación. Si el nombre resulta demasiado largo, corto o críptico, existe la posibilidad de cambiarlo con la misma sentencia SQL de consulta creando un ALIAS. El ALIAS se pone entre comillas dobles, a la derecha de la columna.

```
SELECT DNOMBRE "Departamento", DEPT_NO "Número Departamento"
FROM DEPART;
```

Ejemplo1:

```
SELECT NOMBRE_ALUMNO "Nombre Alumno", (NOTA1+NOTA2+NOTA3)/3
"Nota Media" FROM NOTAS_ALUMNOS;
```

Ejemplo2:

```
SELECT NOMBRE_ALUMNO FROM NOTAS_ALUMNOS WHERE NOTA1=7 AND
(NOTA1+NOTA2+NOTA3)/3 >6;
```

2. Operadores de comparación de cadenas de caracteres

Para comparar cadenas de caracteres, hasta ahora hemos utilizado el operador de comparación Igual a (=). Por ejemplo, a partir de la tabla EMPLE, obtenemos el apellido de los ANALISTAS del departamento 10:

```
SELECT APELLIDO FROM EMPLE WHERE OFICIO = 'ANALISTA' AND
DEPT_NO=10;
```


Pero este operador no nos sirve si queremos hacer consultas de este tipo: Obtener los datos de los empleados cuyo apellido empiece por una «P» u «obtener los nombres de alumnos que incluyan la palabra Pérez». Para especificar este tipo de consultas, en SQL usamos el operador LIKE que permite utilizar los siguientes caracteres especiales en las cadenas de comparación:

- % Comodín: representa cualquier cadena de 0 o más caracteres.
- '_' Marcador de posición: representa un carácter cualquiera.

Ejemplos: (Hemos de tener en cuenta que las mayúsculas y minúsculas son significativas, 'm' no es lo mismo que 'M', y que las constantes alfanuméricas deben encerrarse siempre entre comillas simples).

- LIKE 'Director' la cadena 'Director'.
- LIKE 'M%' cualquier cadena que empiece por 'M'.
- LIKE '%X%' cualquier cadena que contenga una 'X'.
- LIKE '__M' cualquier cadena de 3 caracteres terminada en 'M'.

- LIKE 'N_' una cadena de 2 caracteres que empiece por 'N'.
- LIKE '_R%' cualquier cadena cuyo segundo carácter sea una 'R'.

Ejemplo 1: (apellidos que empiezan por J)

```
SELECT APELLIDO FROM EMPLE WHERE APELLIDO LIKE 'J%';
```

Ejemplo 2: (apellidos que tengan una R en segunda posición)

```
SELECT APELLIDO FROM EMPLE WHERE APELLIDO LIKE '_R%';
```

Ejemplo 3: (apellidos que empieces por A y tengan una O en su interior)

```
SELECT APELLIDO FROM EMPLE WHERE APELLIDO LIKE 'A%O%';
```

3. 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, usamos la expresión: *columna IS NOT NULL*. Cuando comparamos con valores nulos o no nulos no podemos utilizar los operadores de igualdad, mayor o menor.

Por ejemplo, a partir de la tabla EMPLE, consultamos los apellidos de los empleados cuya comisión es nula:

```
SELECT APELLIDO FROM EMPLE WHERE COMISION IS NULL;
```

Si queremos consultar los apellidos de los empleados cuya comisión no sea nula teclearemos esto:


```
SELECT APELLIDO FROM EMPLE WHERE COMISION IS NOT NULL;
```

4. IN y BETWEEN

También podemos comparar una columna o una expresión con una lista de valores utilizando los operadores IN y BETWEEN.

El operador **IN** nos permite comprobar si una expresión pertenece o no (NOT) a un conjunto de valores, haciendo posible la realización de comparaciones múltiples. Su formato es:

```
<expresión> [NOT] IN (lista de valores separados por comas)
```

Ejemplo 1: Consulta los apellidos de la tabla EMPLE cuyo número de departamento sea 10 o 30:

```
SELECT APELLIDO FROM EMPLE WHERE DEPT_NO IN(10,30);
```

Ejemplo 2: Consulta los apellidos de la tabla EMPLE cuyo número de departamento no sea ni 10 ni 30:

```
SELECT APELLIDO FROM EMPLE WHERE DEPT_NO NOT IN(10,30);
```

Ejemplo 3: Consulta los apellidos de la tabla EMPLE cuyo oficio sea 'VENDEDOR', 'ANALISTA' o 'EMPLEADO':

```
SELECT APELLIDO FROM EMPLE WHERE OFICIO IN ('VENDEDOR',
'ANALISTA', 'EMPLEADO');
```

El operador **BETWEEN** comprueba si un valor está comprendido o no (NOT) dentro de un rango de valores, desde un valor inicial a un valor final. Su formato es:

```
<expresión> [NOT] BETWEEN valor_inicial AND valor_final
```

Ejemplo 4: A partir de la tabla EMPLE, obtén el apellido y el salario de los empleados cuyo salario esté comprendido entre 1500 y 2000:

```
SELECT APELLIDO, SALARIO FROM EMPLE WHERE SALARIO BETWEEN
1500 AND 2000;
```

5. Subconsultas

A veces, para realizar alguna operación de consulta, necesitamos los datos devueltos por otra consulta; una subconsulta, que no es más que una sentencia SELECT dentro de otra SELECT. Las subconsultas son aquellas sentencias SELECT que forman parte de una cláusula WHERE de una sentencia SELECT anterior. Una subconsulta consistirá en incluir una declaración SELECT como parte de una cláusula WHERE. El formato de una subconsulta es similar a éste:

```
SELECT ...
FROM ...
WHERE columna operador_comparativo (SELECT ...
                                    FROM ...
                                    WHERE ... );
```


La subconsulta (el comando SELECT entre paréntesis) se ejecutará primero y, posteriormente, el valor extraído es «introducido» en la consulta principal.

Ejemplo 1: Con la tabla EMPLE, obtén el APELLIDO de los empleados con el mismo OFICIO que 'GIL'.

```
SELECT APELLIDO FROM EMPLE WHERE OFICIO = (SELECT OFICIO
FROM EMPLE WHERE APELLIDO ='GIL');
```

Ejemplo 2: Obtener aquellos apellidos de empleados cuyo oficio sea alguno de los oficios que hay en el departamento 20.

```
SELECT APELLIDO FROM EMPLE WHERE OFICIO IN (SELECT OFICIO
FROM EMPLE WHERE DEPT_NO=20);
```

Ejemplo 3: Listar los departamentos que tengan empleados.

```
SELECT DNOMBRE, DEPT_NO FROM DEPART WHERE EXISTS (SELECT
* FROM EMPLE WHERE EMPLE.DEPT_NO= DEPART.DEPT_NO);
```

Ejemplo 4: Obtener los datos de los empleados cuyo salario sea igual a algún salario de los empleados del departamento 30.

```
SELECT * FROM EMPLE WHERE SALARIO = ANY (SELECT SALARIO
FROM EMPLE WHERE DEPT_NO=30);
```

Ejemplo 5: Obtener los datos de los empleados cuyo salario sea menor a cualquier salario de los empleados del departamento 30.

```
SELECT * FROM EMPLE WHERE SALARIO < ALL (SELECT SALARIO
FROM EMPLE WHERE DEPT_NO=30);
```

Ejemplo 6: Usamos las tablas EMPLE y DEPART. Queremos consultar los datos de los empleados que trabajen en 'MADRID' o 'BARCELONA'. La localidad de los departamentos se obtiene de la tabla DEPART. Hemos de relacionar las tablas EMPLE y DEPART por el número de departamento.

```
SELECT EMP_NO, APELLIDO, OFICIO, DIR, FECHA_ALT, SALARIO,
COMISION, DEPT_NO
FROM EMPLE WHERE DEPT_NO IN (SELECT DEPT_NO FROM DEPART WHERE
LOC IN ('MADRID','BARCELONA'));
```

6. Combinación de tablas

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

```
SELECT columnas
FROM tabla1, tabla2,...
WHERE tabla1.columna = tabla2.columna;
```

Ejemplo 1: Realiza una consulta para obtener el nombre de alumno, su asignatura y su nota:

```
SELECT APELLIDOSNOMBRE, NOMBRE_ASIG, NOTA FROM ALUMNOS,
ASIGNATURAS, NOTAS
WHERE ALUMNOS.DNI=NOTAS.DNI AND NOTAS.COD=ASIGNATURAS.COD;
```

Ejemplo 2: Obtén los nombres de alumnos matriculados en 'FOL':

```
SELECT APELLIDOSNOMBRE FROM ALUMNOS, ASIGNATURAS, NOTAS
WHERE ALUMNOS.DNI=NOTAS.DNI AND NOTAS.COD=ASIGNATURAS.COD
AND NOMBRE_ASIG ='FOL';
```

7. Tipos de JOIN

Lo que hemos hecho en el apartado anterior es equivalente a realizar una operación de unión de tablas JOIN.

La operación JOIN o combinación permite mostrar columnas de varias tablas como si se tratase de una sola tabla, combinando entre sí los registros relacionados usando para ello claves externas. Las tablas relacionadas se especifican en la cláusula FROM, y además hay que hacer coincidir los valores que relacionan las columnas de las tablas.

```
SELECT [ ALL / DISTINCT ] [ * ] / [ListaColumnas_Expresiones]
FROM NombreTabla1 JOIN NombreTabla2 ON
Condiciones_Vinculos_Tablas
```

Ejemplo 1: Realiza una consulta para obtener el nombre de alumno, su asignatura y su nota:

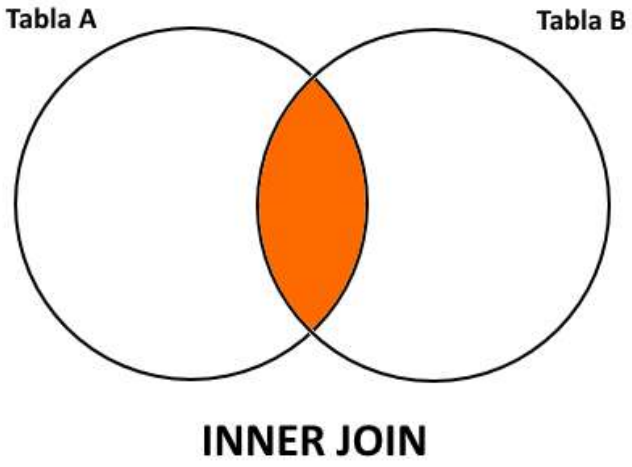
(sin JOIN)

```
SELECT APELLIDOSNOMBRE, NOMBRE_ASIG, NOTA
FROM ALUMNOS AL, ASIGNATURAS AS, NOTAS N
WHERE AL.DNI=N.DNI AND N.COD=AS.COD;
```

(con JOIN)

```
SELECT APELLIDOSNOMBRE, NOMBRE_ASIG, NOTA
FROM ALUMNOS AL JOIN NOTAS N ON AL.DNI=N.DNI
JOIN ASIGNATURAS AS ON N.COD=AS.COD;
```


Las combinaciones internas se realizan mediante la instrucción **INNER JOIN**. Devuelven únicamente aquellos registros/filas que tienen valores idénticos en los dos campos que se comparan para unir ambas tablas. Es decir, aquellas que tienen elementos en las dos tablas, identificados éstos por el campo de relación. Veamos un diagrama que representa esto:



INNER JOIN

En este caso se devuelven los registros que tienen nexo de unión en ambas tablas. En realidad, esto ya lo conocíamos puesto que, en las combinaciones internas, el uso de la palabra INNER es opcional así que si simplemente indicamos la palabra JOIN y la combinación de columnas el sistema sobreentiende que estamos haciendo una combinación interna (INNER JOIN).

Las combinaciones externas se realizan mediante la instrucción **OUTER JOIN**. Devuelven todos los valores de la tabla que hemos puesto a la derecha, los de la tabla que hemos puesto a la izquierda o los de ambas tablas según el caso, devolviendo además valores nulos en las columnas de las tablas que no tengan el valor existente en la otra tabla.

Es decir, que nos permite seleccionar algunas filas de una tabla, aunque éstas no tengan correspondencia con las filas de la otra tabla con la que se combina. Ahora lo veremos mejor en cada caso concreto, ilustrándolo con un diagrama para una mejor comprensión.

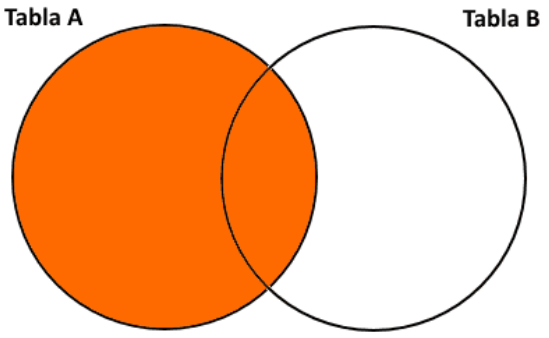
La sintaxis general de las combinaciones externas es:

```
FROM Tabla1 [LEFT/RIGHT/FULL] [OUTER] JOIN Tabla2 ON
Condiciones_Vinculos_Tablas
```

En todas estas combinaciones externas el uso de la palabra OUTER es opcional. Si utilizamos LEFT, RIGHT o FULL y la combinación de columnas, el sistema sobreentiende que estamos haciendo una combinación externa.

LEFT JOIN: Se obtienen todas las filas de la tabla colocada a la izquierda, aunque no tengan correspondencia en la tabla de la derecha:

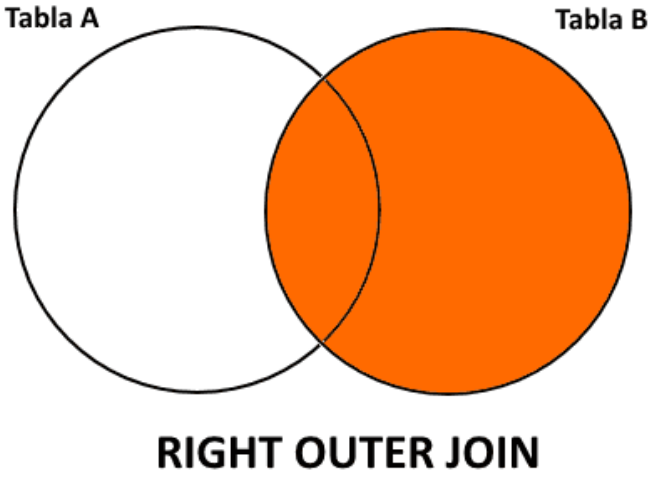
```
SELECT T1.Col1, T1.Col2, T1.Col3, T2.Col7
FROM Tabla1 T1 LEFT [OUTER] JOIN Tabla2 T2 ON T1.Col1 =
T2.Col1
```

LEFT OUTER JOIN

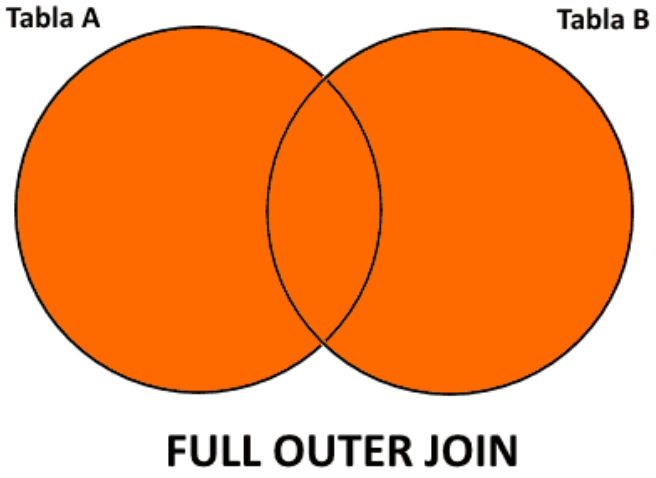
RIGHT JOIN: usando RIGHT JOIN se obtienen todas las filas de la tabla de la derecha, aunque no tengan correspondencia en la tabla de la izquierda.

```
SELECT T1.Col1, T1.Col2, T1.Col3, T2.Col7
FROM Tabla1 T1 RIGHT [OUTER] JOIN Tabla2 T2 ON T1.Col1 =
T2.Col1
```



FULL JOIN: Se obtienen todas las filas en ambas tablas, aunque no tengan correspondencia en la otra tabla. Es decir, todos los registros de A y de B aunque no haya correspondencia entre ellos, rellenando con nulos los campos que falten:

```
SELECT T1.Col1, T1.Col2, T1.Col3, T2.Col7
FROM Tabla1 T1 FULL [OUTER] JOIN Tabla2 T2 ON T1.Col1 =
T2.Col1
```



8. Operadores UNION, INTERSECT y MINUS

Los operadores relacionales UNION, INTERSECT y MINUS son operadores de conjuntos. Los conjuntos son las filas resultantes de cualquier sentencia SELECT válida que permiten combinar los resultados de varias SELECT para obtener un único resultado.

Supongamos que tenemos dos listas de centros de enseñanza de una ciudad y que queremos enviar a esos centros una serie de paquetes de libros. Dependiendo de ciertas características de los centros, podemos enviar libros a todos los centros de ambas listas (UNION), a los centros que estén en las dos listas (INTERSECT) o a los que están en una lista y no están en la otra (MINUS). El formato de SELECT con estos operadores es el siguiente:

```
SELECT ... FROM ... WHERE...
Operador_de_conjunto
SELECT ... FROM ... WHERE...
```

El operador **UNION** combina los resultados de dos consultas. Las filas duplicadas que aparecen se reducen a una fila única. Éste es su formato:

```
SELECT COL1, COL2, ... FROM TABLA1 WHERE CONDICION
UNION
SELECT COL1, COL2, ... FROM TABLA2 WHERE CONDICION;
```

Ejemplo 1:

ALUM contiene los nombres de alumnos que se han matriculado este curso en el centro, NUEVOS contiene los nombres de los alumnos que han reservado plaza para el próximo curso y ANTIGUOS contiene los nombres de antiguos alumnos del centro. Queremos visualizar los nombres de los alumnos actuales y de los futuros alumnos. Obtenemos los nombres de alumnos que aparezcan en las tablas ALUM y NUEVOS de la siguiente manera:

```
SELECT NOMBRE FROM ALUM UNION SELECT NOMBRE FROM NUEVOS;
```

UNION ALL combina los resultados de dos consultas. Cualquier duplicación de filas que se dé en el resultado final aparecerá en la consulta. En la consulta anterior, usando UNION ALL aparecerán nombres duplicados:

```
SELECT NOMBRE FROM ALUM UNION ALL SELECT NOMBRE FROM NUEVOS;
```

El operador **INTERSECT** devuelve las filas que son iguales en ambas consultas. Todas las filas duplicadas serán eliminadas antes de la generación del resultado final. Su formato es:

```
SELECT COL1, COL2, ... FROM TABLA1 WHERE CONDICION
INTERSECT
SELECT COL1, COL2, ... FROM TABLA2 WHERE CONDICION;
```

Ejemplo 2: Obtén los nombres de alumnos que están actualmente en el centro y que estuvieron en el centro hace ya un tiempo. Necesitamos los nombres que están en la tabla ALUM y que, además, aparezcan en la tabla de ANTIGUOS alumnos:

```
SELECT NOMBRE FROM ALUM INTERSECT SELECT NOMBRE FROM
ANTIGUOS;
```

El operador **MINUS** devuelve aquellas filas que están en la primera SELECT y no en la segunda. Las filas duplicadas del primer conjunto se reducirán a una fila única antes de que empiece la comparación con el otro conjunto. Su formato es éste:

```
SELECT COL1, COL2, ... FROM TABLA1 WHERE CONDICION
MINUS
SELECT COL1, COL2, ... FROM TABLA2 WHERE CONDICION;
```

Obtén los nombres y la localidad de alumnos que están actualmente en el centro y que nunca estuvieron anteriormente en él. Ordenamos la salida por LOCALIDAD. Necesitamos los nombres que están en la tabla ALUM y que, además, no aparezcan en la tabla de ANTIGUOS alumnos:

```
SELECT NOMBRE, LOCALIDAD FROM ALUM MINUS
SELECT NOMBRE, LOCALIDAD FROM ANTIGUOS
ORDER BY LOCALIDAD;
```

9. Funciones de agregación

Las funciones de agregación en SQL nos permiten efectuar operaciones sobre un conjunto de resultados, pero devolviendo un único valor agregado para todos ellos. Es decir, nos permiten obtener medias, máximos, etc... sobre un conjunto de valores.

Las funciones de agregación básicas que soportan todos los gestores de datos son las siguientes:

- COUNT: devuelve el número total de filas seleccionadas por la consulta.
- MIN: devuelve el valor mínimo del campo que especifiquemos.
- MAX: devuelve el valor máximo del campo que especifiquemos.
- SUM: suma los valores del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.
- AVG: devuelve el valor promedio del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.

Las funciones anteriores son las básicas en SQL, pero cada sistema gestor de bases de datos relacionales ofrece su propio conjunto, más amplio, con otras funciones de agregación particulares.

Ejemplo 1: Cálculo del salario medio de los empleados del departamento 10 de la tabla EMPLE:

```
SELECT AVG(SALARIO) FROM EMPLE WHERE DEPT_NO=10;
```

Ejemplo 2: Cálculo del máximo salario de la tabla EMPLE:

```
SQL> SELECT MAX(SALARIO) FROM EMPLE;
```

Todas estas funciones se aplican a una sola columna, que especificaremos entre paréntesis, excepto la función COUNT, que se puede aplicar a una columna o indicar un "*". La diferencia entre poner el nombre de una columna o un "*", es que en el primer caso no cuenta los valores nulos para dicha columna, y en el segundo sí.

Ejemplo 3: Cálculo del número de filas de la tabla EMPLE:

```
SELECT COUNT(*) FROM EMPLE;
```

Ejemplo 4: Cálculo del número de filas de la tabla EMPLE donde la COMISION no es nula:

```
SELECT COUNT(COMISION) FROM EMPLE;
```

Ejemplo 5: Calcula el número de oficios que hay en la tabla EMPLE:

```
SELECT COUNT(OFICIO) "OFICIOS" FROM EMPLE;
```

Esta consulta cuenta todos los oficios de la tabla EMPLE que no sean nulos, estén repetidos o no. Si queremos contar los distintos oficios que hay en la tabla EMPLE, tendríamos que incluir DISTINCT en la función de grupo:

```
SELECT COUNT(DISTINCT OFICIO) "OFICIOS" FROM EMPLE;
```

10. Agrupando resultados (GROUP BY y HAVING)

La cláusula GROUP BY unida a un SELECT permite agrupar filas según las columnas que se indiquen como parámetros, y se suele utilizar en conjunto con las funciones de agrupación, para obtener datos resumidos y agrupados por las columnas que se necesiten.

Por ejemplo: Para saber cuál es el salario medio de cada departamento de la tabla EMPLE necesitamos realizar un agrupamiento por departamento. Para ello utilizaremos la cláusula GROUP BY. La consulta sería la siguiente:

```
SELECT DEPT_NO, AVG(SALARIO) FROM EMPLE GROUP BY DEPT_NO;
```

La sentencia SELECT posibilita agrupar uno o más conjuntos de filas. El agrupamiento se lleva a cabo mediante la cláusula GROUP BY por las columnas especificadas y en el orden especificado. Éste es su formato:

```
SELECT ...
FROM ...
GROUP BY columna1, columna2, columna3,...
HAVING condición
ORDER BY ...
```

Los datos seleccionados en la sentencia SELECT que lleva el GROUP BY deben ser: una constante, una función de grupo (SUM, COUNT, AVG, ...), una columna expresada en el GROUP BY.

La cláusula GROUP BY sirve para calcular propiedades de uno o más conjuntos de filas. Además, si se selecciona más de un conjunto de filas, GROUP BY controla que las filas de la tabla original sean agrupadas en una temporal. Del mismo modo que existe la condición de búsqueda WHERE para filas individuales, también hay una condición de búsqueda para grupos de filas: HAVING. La cláusula HAVING se emplea para controlar cuál de los conjuntos de filas se visualiza. Se evalúa sobre la tabla que devuelve el GROUP BY. No puede existir sin GROUP BY.

Ejemplo 1: Visualiza a partir de la tabla EMPLE el número de empleados que hay en cada departamento. Para hacer esta consulta, tenemos que agrupar las filas de la tabla EMPLE por departamento (GROUP BY DEPT_NO) y contarlas (COUNT(*)). La consulta es la siguiente:

```
SELECT DEPT_NO, COUNT(*) FROM EMPLE GROUP BY DEPT_NO;
```

COUNT es una función de grupo y da información sobre un grupo de filas, no sobre filas individuales de la tabla. La cláusula GROUP BY DEPT_NO obliga a COUNT a contar las filas que se han agrupado por cada departamento.

Ejemplo 2: Si en la consulta anterior sólo queremos visualizar los departamentos con más de 4 empleados, tendríamos que escribir lo siguiente:

```
SELECT DEPT_NO, COUNT(*) FROM EMPLE GROUP BY DEPT_NO HAVING
COUNT(*) > 4;
```

11. Funciones en consultas de selección

En el lenguaje SQL estándar existen básicamente 5 tipos de funciones: aritméticas, de cadenas de caracteres, de fechas, de conversión, y otras funciones diversas que no se pueden incluir en ninguno de los grupos anteriores.

Es muy importante tener en cuenta que, habitualmente, cualquier sistema gestor de bases de datos relacionales (SGBDR) intenta incluir la mayoría de las funciones correspondientes al estándar ANSI SQL, y que además suelen incluir un conjunto adicional de funciones propias. Incluso de una versión a la siguiente dentro de un mismo producto, es común que aparezcan nuevas funciones. Aquí nos vamos a limitar a citar algunas de las más significativas, incluyendo una breve descripción de las mismas:

Funciones aritméticas:

- **ABS(n)**: Devuelve el valor absoluto de “n”.
- **ROUND(m, n)**: Redondea el número “m” con el número de decimales indicado en “n”, si no se indica “n” asume cero decimales.
- **SQRT(n)**: Devuelve la raíz cuadrada del parámetro que se le pase.
- **POWER(m, n)**: Devuelve la potencia de “m” elevada el exponente “n”.

Funciones de cadenas

- **LOWER(c)**: Devuelve la cadena "c" con todas las letras convertidas a minúsculas.
- **UPPER(c)**: Devuelve la cadena "c" con todas las letras convertidas a mayúsculas.
- **LTRIM(c)**: Elimina los espacios por la izquierda de la cadena "c".
- **RTRIM(c)**: Elimina los espacios por la derecha de la cadena "c".
- **REPLACE(c, b, s)**: Sustituye en la cadena "c" el valor buscado "b" por el valor indicado en "s".
- **REPLICATE(c, n)**: Devuelve el valor de la cadena "c" el número de veces "n" indicado.
- **LEFT(c, n)**: Devuelve "n" caracteres por la izquierda de la cadena "c".
- **RIGHT(c, n)**: Devuelve "n" caracteres por la derecha de la cadena "c".
- **SUBSTRING(c, m, n)**: Devuelve una sub-cadena obtenida de la cadena "c", a partir de la posición "m" y tomando "n" caracteres.

Funciones de manejo de fechas

- **YEAR(d)**: Devuelve el año correspondiente de la fecha "d".
- **MONTH(d)**: Devuelve el mes de la fecha "d".
- **DAY(d)**: Devuelve el día del mes de la fecha "d".
- **DATEADD(f, n, d)**: Devuelve una fecha "n" periodos (días, meses, años, según lo indicado) superior a la fecha "d". Si se le pasa un número "n" negativo, devuelve una fecha "n" periodos inferior.

Funciones de conversión

Estas funciones **suelen ser específicas de cada gestor de datos**, ya que cada SGBDR utiliza nombres diferentes para los distintos tipos de datos. Las funciones de conversión nos permiten cambiar valores de un tipo de datos a otro. Por ejemplo, si tenemos una cadena y sabemos que contiene una fecha, podemos convertirla al tipo de datos fecha. Así, por ejemplo:

- En **SQL Server** tenemos funciones como CAST, CONVERT o PARSE, que en función de lo que especifiquemos en sus parámetros convertirán los datos en el tipo que le indiquemos.
- En **Oracle** tenemos funciones como TO_CHAR, TO_DATE, TO_NUMBER.
- En **MySQL** solucionamos la mayor parte de las conversiones con la función CAST.

12. Inserción de datos con SELECT

En el manual anterior de SQL aprendimos a insertar datos en las tablas directamente. Pero también existe la posibilidad de añadir a INSERT una consulta, es decir, una sentencia SELECT, se añaden tantas filas como devuelva la consulta. El formato de INSERT con SELECT es el siguiente:

```
INSERT INTO NombreTabla1 [(columna [, columna] ...)]
SELECT {columna [, columna] ... | *}
FROM NombreTabla2 [CLÁUSULAS DE SELECT];
```

Ejemplo 1:

```
INSERT INTO EMPLE30 SELECT * FROM EMPLE WHERE DEPT_NO=30;
```

Ejemplo 2:

```
INSERT INTO EMPLE30
EMP_NO, APELLIDO, OFICIO, DIR, FECHA_ALT, SALARIO, COMISION,
DEPT_NO)
SELECT
EMP_NO, APELLIDO, OFICIO, DIR, FECHA_ALT, SALARIO, COMISION,
DEPT_NO
FROM EMPLE
WHERE DEPT_NO=30;
```

13. Modificación de datos con SELECT

Podemos incluir una subconsulta en una sentencia UPDATE que puede estar contenida en la cláusula WHERE o puede formar parte de SET. Cuando la subconsulta (orden SELECT) forma parte de SET, debe seleccionar una única fila y el mismo número de columnas (con tipos de datos adecuados) que las que hay entre paréntesis al lado de SET. Los formatos son:

```
UPDATE <NombreTabla>
SET columna1 = valor1, columna2 = valor2, ...
WHERE columna3 = (SELECT ...);

UPDATE <NombreTabla>
SET (columna1, columna2, ...) = (SELECT col1, col2, ...)
WHERE condición;

UPDATE <NombreTabla>
SET columna1 = (SELECT col1 ...), columna2 = (SELECT
col2 ...)
WHERE condición;
```

Ejemplo 1: En la tabla CENTROS la siguiente orden UPDATE igualará la dirección y el número de plazas del código de centro 10 a los valores de las columnas correspondientes que están almacenadas para el código de centro 50. Los valores actuales de estos centros son:

```
UPDATE CENTROS SET (DIRECCION, NUM_PLAZAS) = (SELECT
DIRECCION, NUM_PLAZAS FROM CENTROS WHERE COD_CENTRO = 50)
WHERE COD_CENTRO= 10;
```

Ejemplo 2: A partir de la tabla EMPLE, cambia el salario a la mitad y la comisión a 0, a aquellos empleados que pertenezcan al departamento con mayor número de empleados.

```
UPDATE EMPLE SET SALARIO = SALARIO/2, COMISION = 0 WHERE
DEPT_NO = (SELECT DEPT_NO FROM EMPLE GROUP BY DEPT_NO HAVING
COUNT(*) = (SELECT MAX(COUNT(*)) FROM EMPLE GROUP BY
DEPT_NO));
```

Ejemplo 3: Para todos los empleados de la tabla EMPLE y del departamento de 'CONTABILIDAD', cambiamos su salario al doble del salario de 'SÁNCHEZ' y su apellido, a minúscula.

```
UPDATE EMPLE SET APELLIDO = LOWER(APELLIDO),
SALARIO = (SELECT SALARIO*2 FROM EMPLE WHERE APELLIDO =
'SANCHEZ')
WHERE DEPT_NO = (SELECT DEPT_NO FROM DEPART
WHERE DNOMBRE = 'CONTABILIDAD');
```


14. Borrado de datos con SELECT

Al igual que podemos incluir una subconsulta en una sentencia INSERT y UPDATE se puede hacer en una sentencia DELETE.

Ejemplo 1: Borramos todas las filas de la tabla LIBRERIA cuyos EJEMPLARES no superen la media de ejemplares en su ESTANTE.

```
DELETE FROM LIBRERIA L WHERE EJEMPLARES <
(SELECT AVG(EJEMPLARES) FROM LIBRERIA WHERE ESTANTE =
L.ESTANTE GROUP BY ESTANTE);
```

Ejemplo 2: Borramos los departamentos de la tabla DEPART con menos de cuatro empleados.

```
DELETE FROM DEPART WHERE DEPT_NO IN
(SELECT DEPT_NO FROM EMPLE GROUP BY DEPT_NO HAVING COUNT(*)
< 4);
```

15. Creación de una tabla con datos de un SELECT

La sentencia CREATE TABLE permite crear una tabla a partir de la consulta de otra tabla ya existente. La nueva tabla contendrá los datos obtenidos en la consulta. Se lleva a cabo esta acción con la cláusula AS colocada al final de la orden CREATE TABLE. El formato es el que sigue:

```
CREATE TABLE Nombretabla
(
Columna [, Columna]
)
AS consulta;
```

No es necesario especificar tipos ni tamaño de las columnas, ya que vienen determinados por los tipos y los tamaños de las recuperadas en la consulta. La consulta puede contener una subconsulta, una combinación de tablas o cualquier sentencia SELECT válida. Las restricciones CON NOMBRE no se crean en una tabla desde la otra, sólo se crean aquellas restricciones que carecen de nombre.

Ejemplo: Se crea la tabla EMPLEYDEPART a partir de las tablas EMPLE y DEPART. Esta tabla contendrá el apellido y el nombre del departamento de cada empleado:

```
CREATE TABLE EMPLEYDEPART
AS SELECT APELLIDO, DNOMBRE
FROM EMPLE, DEPART WHERE EMPLE.DEPT_NO = DEPART.DEPT_NO;
```

16. Índices.

Sin un índice, el sistema de base de datos lee a través de toda la tabla (este proceso se denomina "escaneo de tabla") para localizar la información deseada. Con el índice correcto en su lugar, el sistema de base de datos puede entonces primero dirigirse al índice para encontrar de dónde obtener los datos, y luego dirigirse a dichas ubicaciones para obtener los datos necesarios. Esto es mucho más rápido.

Por lo tanto, generalmente se recomienda crear índices en tablas. Un índice puede cubrir una o más columnas. No es aconsejable que se utilicen índices en campos de tablas pequeñas o campos que se actualicen con mucha frecuencia.

El diseño de índices es un tema bastante complejo para los Administradores de Bases de Datos, ya que una mala elección ocasiona ineficiencia y tiempos de espera elevados. Un uso excesivo de ellos puede dejar a la Base de Datos colgada simplemente con insertar alguna fila. Para crear un índice utilizaríamos la siguiente sentencia:

```
CREATE INDEX NombreIndice ON NombreTabla (Columna1 [,
Columna2 ...]);
```

Para eliminar un índice es suficiente con poner la instrucción:

```
DROP INDEX NombreIndice;
```

Ejemplo: En una tabla de datos personales llamada “usuarios” podemos hacer un índice sobre la columna “apellidos”.

```
CREATE INDEX idx_apellidos ON usuarios(apellidos);
```

17. Vistas.

Las vistas permiten definir subconjuntos de datos formados con una o varias tablas y/o vistas. Tiene la apariencia de una tabla, pero ocupa menos espacio que éstas ya que solo se almacena la definición de la vista y no los datos que la forman. Gracias a las vistas podemos mostrar a los usuarios una visión parcial de los datos, de modo que podamos mostrar a los usuarios sólo aquellos datos que son de su interés.

Cuando creamos una vista lo que hacemos es aplicar una consulta sobre la base de datos. La diferencia es que la vista se queda almacenada y podrá ser reutilizada en el futuro. Para crear una vista usamos:

```
CREATE VIEW NombreVista AS Subconsulta;
```

Las vistas también podrán ser eliminadas de la base de datos. Para ello escribimos lo siguiente:

```
DROP VIEW NombreVista;
```

Ejemplo: En la tabla personas

nombre	apellido1	apellido2	edad
ANTONIO	PEREZ	GOMEZ	30
ANTONIO	GARCIA	RODRIGUEZ	45
PEDRO	RUIZ	GONZALEZ	50

Creamos una vista con las personas que se llaman ANTONIO

```
CREATE VIEW antonios AS
SELECT nombre, apellido1, apellido2, edad
FROM personas
WHERE nombre = 'ANTONIO';
```

Para consultar los datos de una vista

```
SELECT * FROM antonios;
```

nombre	apellido1	apellido2	edad
ANTONIO	PEREZ	GOMEZ	30
ANTONIO	GARCIA	RODRIGUEZ	45