LENGUAJE DE MANIPULACIÓN DE DATOS(LMD)

El LMD está formado por cuatro sentencias, que proporcionan el acceso al contenido de la base de datos, no modificando la estructura de la misma, estas son:

- Sentencia SELECT. Consulta el contenido de la base de datos, no produciéndose modificación alguna.
- Sentencia INSERT. Inserta filas dentro de tablas de la base de datos.
- Sentencia UPDATE. Modifica el contenido de las tablas en la base de datos
- Sentencia DELETE. Borra filas dentro de las tablas de la base de datos.

Las tres últimas sentencias modifican el contenido de las tablas de la base de datos, en ningún caso la estructura de la misma.

1.- OPERACIÓN DE RECUPERACIÓN.-

La principal operación es la de Transformación, que se representa sintácticamente como un bloque SELECT-FROM-WHERE.

La consulta es relacionalmente completa, porque puede obtenerse:

- Cualquier unidad de datos.
- Cualquier subconjunto de datos (filas y columnas)

Los datos se pueden representar en un orden determinado.

SELECT {*}{<Nombre_columna>[,< Nombre_columna>...]} (3)
FROM <Nombre_tabla>[,<Nombre_tabla>...] (1)
[WHERE <predicado>] (2)
[ORDER BY <columna>[{ASC}{DESC}] [,<columna>[{ASC}{DESC}]…] (4)

Entre paréntesis se encuentra el orden en que se ejecutarán las sentencias.

- SELECT.- Sirve para elegir, ciertas columnas o valores derivados de ellas.
- FROM.- Especifica la lista de tablas que se utilizarán en la consulta. Se recuerda que el nombre de la tabla está formado por nombre_propietario.nombre_tabla. Si el dueño de la tabla es el usuario conectado, podrá omitirse, colocando solamente el nombre de la tabla.
- WHERE.- Selecciona aquellas filas cuyos campos cumplen la condición o condiciones indicadas en el predicado.
- ORDER BY.- Especifica el criterio de ordenación, basado en los valores de los datos o de otros derivados de ellos. Puede indicar el nombre de la columna, el número de orden correspondiente a la misma en la select, el alias de la columna, incluso ordenar por un campo que se encuentre en la tabla, pero no aparezca en la select, excepto si se utilizó la cláusula distinct .

La lista de las columnas extraídas en la select, viene encabezada por el nombre de la columna. Si se desea eliminar este, o en caso de valores derivados de las mismas, a continuación del nombre del campo, se colocará el "Alias", irá precedido por la palabra AS o sencillamente separado por " " y entrecomillado("") de forma obligatoria en caso de contener blancos y opcional en caso contrario.

Tomaremos como referencia la base de datos de proveedores, artículos, proyectos y ventas (s,p,j,spj).

Recuperación Simple.
1.-**Obtener los números de parte, de todas las partes suministradas.**

Debemos tener en cuenta, que las partes suministradas, son las partes vendidas y que por tanto estarán en SPJ.

Select P# from SPJ.

Como se puede comprobar, en la solución aparecen demasiados duplicados, uno por cada venta realizada por cada uno de los proveedores. Para eliminar filas duplicadas, utilizaremos la palabra clave DISTINCT. Por tanto lo correcto sería:

```
select distinct p# from spj
```

2.- Obtener todos los detalles de todos los proveedores.

Esta consulta tiene dos posibles soluciones, igualmente óptimas:

a)

```
SELECT *
FROM S
```

b)

```
SELECT S#,NOMS,ESTADO,CIUDAD
FROM S
```

En el primer caso, se utiliza la abreviatura *, que corresponde a todas las columnas de la tabla, en el orden en que fueron insertadas, en el segundo caso, se numeran las mismas de forma explícita.

A efectos de optimización, se recomienda que las sentencias se escriban de la misma forma, pues el motor de la base de datos, guarda las sentencias más utilizadas y la forma de ejecución por tanto resolverá el problema más rápidamente, y para ello, la sintaxis debe ser la misma, se decir select * y select *, ofrecen la misma salida, pero al no estar escritas exactamente igual, el gestor considera que son dos sentencias distintas y por tanto aunque acabe de resolver la primera, tiene que hacer lo mismo con la segunda, si se hubiesen escrito igual, ya "sabría cómo hacerlo".

Recuperación Cualificada.

Obtenga todos los números de los proveedores de París, con estado mayor que 20.
(hay que hacer la observación que sql Server, no distingue entre literales escritos en mayúsculas o minúsculas, pero no en todos los gestores ocurre esto)

```
Select s#
From  s
Where ciudad='PARIS' AND ESTADO>20
```

Sería lo mismo preguntar por 'paris' o 'Paris', pero mucho cuidado, en este gestor en otros no.

La condición o predicado que sigue a la where, puede incluir operadores relacionales, que son =,<,>,>=,<=,!=, y los operadores boléanos AND, OR, NOT; y paréntesis para indicar el orden deseado en la condición. La prioridad de los operadores es la de siempre, es decir, primero sumas y restas, multiplicación y división, concatenación (+), relaciones, negación, conjunción y diyunción.

Recuperación con ordenamiento: Obtenga los números de proveedor y estado de los proveedores de parís, en orden ascendente de estado:

```
select s#,estado
from s
where ciudad='paris' – sería igual 'PARIS'
order by estado Desc
```

Si se desea la ordenación asc, puede omitirse pues se considera por defecto y en lugar del nombre del campo, puede colocarse el número de orden que hace en la select, es decir sería lo mismo ... order by 2 desc.

MÚLTIPLES TABLAS: UNIONES (JOINS) Y SUBCONSULTAS(SUBSELECT)

JOINS

Cuando necesitamos obtener datos que se encuentran en más de una tabla, utilizaremos el concepto de join. Este consiste en realizar el producto cartesiano entre las tablas que intervienen utilizando como enlace el valor del campo común para ambas, y obtener posteriormente un subconjunto del producto cartesiano que se ha formado.

En principio, el orden de colocación de las tablas en la from es arbitrario, pues como ya hemos visto al definir relación, en nuestro caso, el producto cartesiano es conmutativo, aunque insisto, matemáticamente no lo es. Pues bien, a efectos de optimización, se recomienda que la tabla más pequeña, sea la última, pues el sistema es la única que lee entera indexando todas las demás.

Recuperación con más de una tabla (JOIN NORMAL)

```
select distinct p#,ciudad
from spj,s
where s.s#=spj.s#
```

P#	Ciudad
P1	Atenas
P1	Paris
P2	Atenas
P3	Atenas
P3	Londres
P3	Paris
P4	Atenas
P4	Paris
P5	Atenas
P5	Londres
P6	Atenas
P6	Londres

En este ejemplo, se muestra la forma de expresar la unión de varias tablas en SQL. El usuario puede dar nombre a varias tablas en la cláusula FROM, y utilizar los nombres de las mismas en la Select y el la Where, con el fin de resolver ambigüedades, y en algunos casos por motivos de claridad. La cláusula select se podría haber escrito de la siguiente forma, que sería más descriptiva: SELECT DISTINCT SPJ.P#, S.CIUDAD, aunque no es necesario, puesto que p#, solamente está en una tabla lo mismo que ciudad.

La forma de operar de la proposición Select, es la siguiente: Realiza el producto cartesiano entre spj y s, eliminando aquellos renglones donde spj.s# sea distinto de s.s#, y después se extraen solamente las columnas s# y ciudad.

Esta forma de obtener el join, es genérica para cualquier gestor que utilicemos, pero Microsoft (Access, SQL Server...) poseen otra forma de expresar el join dentro de la from, en lugar de en la where, especificando el tipo de combinación, en este caso INNER, por tanto la select anterior quedaría como sigue:

```
select distinct p#,ciudad
from spj inner join s on (s.s#=spj.s#)
```

JOIN A LA IZQUIERDA (LEFT JOIN)
Especifica que se obtengan todas las filas de la tabla situada a la izquierda que no cumplan la condición y todas las de la derecha que la cumplan, las correspondientes a las de la izquierda que no cumplen la condición se rellanan con NULL.

Obtener el nombre de todas las pates y las partes vendidas (cuando se haya vendido)
Para explicar tanto el join a derecha y a la izquierda, vamos a insertar un vendedor que no venda nada, pues da la casualidad de que todos los vendedores han vendido alguna parta, posteriormente se borrará la fila.

Insertamos : 'P7','garcia',10,'Paris'

```
SELECT DISTINCT NOMS,P#
FROM S LEFT JOIN SPJ ON (S.S#=SPJ.S#)
```

La salida es:

Noms	P#
Aldana	P1
Aldana	P2
Aldana	P3
Aldana	P4
Aldana	P5
Aldana	P6
Bernal	P3
Bernal	P4
Caicedo	P6
Garcia	NULL
Jaramillo	P1
Salazar	P3
Salazar	P5

Como se puede apreciar, aparece que la parte vendida por García es NULL, es decir, no ha vendido nada

En caso de duda consulta la ayuda o los libros on line, viene muy claro y la sintaxis es fácil de olvidar

JOIN A LA DERECHA (RIGHT JOIN)

Especifica todas las filas de la tabla de la derecha, aunque no cumplan la condición, más las de la izquierda que si la cumplen, en campo correspondiente se rellena a NULL.

La misma consulta de antes sería:

```
SELECT DISTINCT NOMS,P#
FROM SPJ RIGHT JOIN S ON (S.S#=SPJ.S#)
```

La salida sera la misma de antes.

Recuperación que implica la reunión de una tabla consigo mima.

Obtenga todas las parejas de números de proveedor, tales que los dos proveedores se encuentren en la misma ciudad.

```
select  primero.s#,segundo.s#
from s primero, s segundo
where primero.ciudad=segundo.ciudad
```

S#	S#
S1	S1
S3	S1
S2	S2
S4	S2
S1	S3
S3	S3
S2	S4
S4	S4
S5	S5

como se puede observar, s1 vive en la misma ciudad que él mismo. Vamos a eliminar esta situación, una solución puede ser la siguiente:

```
select primero.s#,segundo.s#
from s primero, s segundo
where primero.ciudad=segundo.ciudad
      and primero.s#!=segundo.s#
```

S#	S#
S3	S1
S4	S2
S1	S3
S2	S4

Sin embargo se observa que los pares se repiten y esto no se resuelve colocando un distinct pues son pares distintos, la solución sería:

```
select  primero.s#,segundo.s#                S#    S#
from s primero, s segundo
where primero.ciudad=segundo.ciudad          S1    S3
    and primero.s#<segundo.s#                S2    S4
```

En este caso, la tabla S se utiliza dos veces, y por tanto, el gestor, carga la misma tabla dos veces en memoria, siendo preciso renombrarla, para "distinguir", la primera de la segunda, en este caso es absolutamente obligatorio el renombrado de al menos una de las dos, en otros casos, renombraremos (daremos un *alias)* las tablas, con el fin de acortar el nombre de la misma.

SUBCONSULTAS

Obtener el nombre de los proveedores que venden la parte P2.

```
SELECT DISTINCT NOMS                 NOMS
FROM SPJ,S                           ALDANA
WHERE P#='P2' AND S.S#=SPJ.S#
```

Se necesita el distinct, porque en otro caso se repetiría Aldana por cada una de las ventas que este ha realizado.

Debemos fijarnos en un dato importante: El dato que se requiere, se encuentra únicamente en la tabla S, aunque se necesita la tabla SPJ para obtener las ventas, es decir la consulta podría ser:

```
SELECT NOMS
FROM S
WHERE S# condición que involucra a 'P2'
```

¿Cuál es la condición que involucra a 'P2'? sencillamente sería obtener aquellos proveedores que venden la parte P2, es decir la consulta:

```
SELECT S#
FROM SPJ
WHERE P#='P2'
```

El resultado de esta consulta, sería el conjunto de todos los proveedores que venden esta parte, en nuestro caso {S5}, y por tanto obtendríamos todos aquellos proveedores cuyo número perteneciera a dicho conjunto, esto es lo que hace el operador IN. Por tanto la consulta sería:

```
SELECT NOMS FROM S WHERE S# IN ('S5')
```

Si sustituimos S5, por la consulta desde la cual se obtuvo, la consulta resultante sería:

```
SELECT NOMS
FROM S
WHERE S# IN (SELECT S#
                  FROM S
                  WHRE P#=’P2’)
```

El resultado es el mismo que antes, y a la consulta que se encuentra entre paréntesis es lo que se conoce con el nombre de **SUBSELECT O SUBCONSULTA**.

MUY IMPORTANTE: CASI SIEMPRE(NO SIEMPRE), UNA CONSULTA QUE SE PUEDE RESOLVER CON UNA SUBSELECT, TAMBIÉN PUEDE HACERSE CON UN JOIN, PERO TEN EN CUENTA QUE: *EL JOIN HACE UN PRODUCTO CARTESIANO EN MEMORIA Y POR TANTO MULTIPLICA, Y LA SUBSELECT CARGA LAS TABLAS EN MEMORIA, POR TANTO SUMA, ASI QUE LA OCUPACIÓN EN MEMORIA DE LA SUBSELECT ES MENOR QUE LA DEL JOIN, SIENDO POR TANTO* ***MÁS ÓPTIMO EL USO DE SUBSELECT QUE DE JOIN, SIEMPRE QUE ESTO SEA POSIBLE***.

Recuperación con varios niveles de anidamiento. Obtener los nombres de aquellos proveedores que suministran al menos una parte roja.

```
select noms
from s
where s# in
            (select s#
             from spj
             where p# in
                        (select p#
                         from p
                         where color='rojo'))
```

NOMS

Jaramillo
Bernal
Caicedo
Aldana

Las subconsultas se pueden anidar en cualquier profundidad.

La lógica la podemos entender resolviendo las subconsultas de abajo a arriba. Primero obtenemos el numero de las partes rojas, después el número de proveedores que las vende y por último como se llaman. No hace distinct, puesto que aunque noms no es clave primaria, si el nombre estuviera duplicado, serían dos proveedores distintos que tienen el mismo nombre.

Recuperación con una subconsulta con referencia entre bloques. (Este tipo de consulta es más difícil de ver, porque no se pueden aislar las subselect, la forma de entenderlas es extraer la tabla dos veces e ir observando el funcionamiento a mano(*).)

La pregunta es la de siempre: obtener el nombre de los proveedores que venden P2.

```
select noms
from s
where 'p2' in (select p#
               from spj
               where s#=s.s#)
```

NOMS
Aldana

en la where de la subconsulta, no hace falta indicar que s# es de la tabla spj, pues el gestor considera por defecto que se trata de la tabla que hay en la select en la cual se encuentra. Si se pone es igualmente correcto y puede que quede más claro.

Recuperación con una subconsulta, con la misma tabla en ambos bloques.

Obtener los números de los proveedores, para aquellos que venden al menos una parte vendida por S2.

```
select distinct s#
from spj
where p# in (
             select p#
             from spj
             where s#='s2')
```

S#
S2
S3
S5

Recuperación con una subconsulta, con referencia entre bloques y la misma tabla en ambos bloques. (en este caso, es necesario el renombrado de las tablas, y para entender el proceso, se recomienda hacer lo mismo que en la consulta de referencia entre bloques (*)).

Obtener el número de parte, de todas aquellas partes vendidas por más de un proveedor.

```
select distinct p#
from spj spx
where p# in (select p#
             from spj
             where s#!=spx.s#)
```

P#
P1
P3
P4
P5
P6

Recuperación usando NOT IN. Obtener los nombres de aquellos proveedores que no venden la parte P2.

Vamos a resolver la consulta de dos formas, obteniendo ambas el mismo resultado y siendo igual de óptimas. Esta sería una consulta que no se podría resolver con un join).

```
select noms
from s
where s# not in (select s#
                 from spj
                 where p#='p2')
```

NOMS

Jaramillo
Salazar
Bernal
Caicedo

```
select noms
from s
where 'p2' not in (select p#
                   from spj
                   where s.s#=s#)
```

LOS OPERADORES EXISTS Y NOT EXISTS LOS VEREMOS AL FINAL

Recuperación usando UNION. Obtenga el número de parte para las partes que pesen más de 18 libras o que actualmente sean suministradas por S2.

(en algunos casos el operador unión se podrá sustituir por la diyunción, pero habrá otros en que no sea posible o sea más óptimo para evitar un join)

El operador unión según la teoría de conjuntos relacional, obtendrá todas las filas de la primera select, mas las de la segunda que no están en la primera, eliminando por tanto duplicados. Los campos seleccionados en cada subselect, serán los mismos y del mismo tipo. Si se deseara ordenación de los datos, aparecerá una sola vez, al final.

UNION es el operador relacional de la teoría de conjuntos relacional, definido como: siendo A y B dos conjuntos no necesariamente disjuntos, se define A U B como el conjunto de elementos x tales que x€A o x€B, o a Ambos. Se eliminan los duplicados.

```
select p#
from p
where peso > 18

union

select p#
from spj
where s#='s2'

order by 1
```

P#

P3
P5
P6

Recuperación de valores computados. Obtener el número de la parte, y el peso de la misma en gramos, para todas las partes, teniendo en cuenta que el peso en la tabla partes está en libras, y que una libra tiene 454 gramos.

Tanto la select como la where, pueden incluir operaciones matemáticas, que comprendan varios campos y/o valores numéricos.

Las columnas en la select, pueden renombrarse (asignarle un alias), se será el valor que aparezca en la salida, éste irá a continuación, sin coma y debe ir entre comilla doble si tuviera blancos, a esto al igual que el renombrado de las tablas en la from se denomina "alias", pues internamente la columna continua con su nombre y si es un campo calculado con ninguno.

```
select p#,peso*454
from p
```

P#	(sin nombre de columna)
P1	5448
P2	7718
P3	7718
P4	6356
P5	5448
P6	8626

Pero por cuestiones de claridad es mejor renombrar la columna del peso:

```
select p#,peso*454 "peso en gramos"
from p
```

P#	peso en gramos
P1	5448
P2	7718
P3	7718
P4	6356
P5	5448
P6	8626

Recuperación que comprenden a NULL. En este caso, vamos a utilizar la bese de datos de dep2 y emp2, que tiene valores nulos.

Si deseamos obtener el nombre de los empleados que no trabajan a comisión, la pregunta a realizar es que deseamos obtener los empleados cuya comisión es nula, es decir la select sería:

```
select apellido,comision
from emp2
where comision is null
```

(en este gestor aparece el valor NULL, en otros este campo queda en blanco).

APELLIDO	COMISION
SANCHEZ	NULL
JIMENEZ	NULL
NEGRO	NULL
CEREZO	NULL
GIL	NULL
REYN	NULL
ALONSO	NULL
JIMENO	NULL
FERNANDEZ	NULL
MUÑOZ	NULL

Obtener los empleados que trabajan a comisión, esto se traduciría como aquellos empleados cuya comisión es no nula, y la select sería:

```
select apellido,comision
from emp2
where comision is null
```

(hay que fijarse que Tovar, tiene comisión aunque esta sea de 0€)

APELLIDO	COMISION
ARROYO	234.00
SALA	391.00
MARTIN	109.00
TOVAR	0.00

Cualquier valor comparado u operado con NULL es NULL.

Las bases de datos que vamos a utilizar a continuación serán las de emp2_dep2 y hospitales, por ser más propicias para la explicación de los casos.

COMPARACIÓN DE CADENA DE CARACTERES. [NOT] LIKE

Es utilizado exclusivamente para comparar cadena de caracteres o fechas(esto en SQL SERVER), permitiendo el uso del carácter comodín.

Comodín	Significado
%	Cualquier cadena de cero o más caracteres.
_	Cualquier carácter.
[ ]	Cualquier carácter individual del intervalo (por ejemplo, [a-f]) o conjunto (por ejemplo, [abcdef]) especificado.
[^]	Cualquier carácter individual fuera del intervalo (por ejemplo, [^a - f]) o conjunto (por ejemplo, [^abcdef]) especificado.

Para escapar los caracteres comodín se utiliza \, al igual que en C. Para más aclaraciones busca en los libros en línea o en la ayuda.

Se utiliza en la where como condición.

COMPARACIÓN LÓGICA

Especifica la comparación (BETWEEN) o (NOT BETWEEN) de un intervalo a otro. La sintaxis es: expresion [NOT] BETWEEN expresion1 AND expresion2, los valores deben ser compatibles. (para más información consulta en la ayuda).

FUNCIONES

IREMOS NUMERANDO LAS FUNCIONES MAS UTILIZADAS, PERO EXISTEN MUCHAS Y LA SINTAXIS LO MAS CONVENIENTE ES CONSULTARLA EN LA AYUDA DEL GESTOR.

FUNCIONES MATEMÁTICAS.
Su argumento es numérico y devuelven un valor numérico.

ABS(n).- Devuelve el valor absoluto de n.

MOD(n,m).- en algunos gestores, devuelve el resto de dividir n entre m, en el caso de SQL Server, se utiliza el operador %, por tanto el reto de dividir n entre m sería n%m.

CEILING(n).- muestra el primer entero igual o superior a n.

FLOOR(n).- muestra el primer entero igual o inferior a n.

EXP(n).- Devuelve n en notación científica y por tanto de tipo float.

POWEER(m,n).- m^n. El valor de n, puede ser positivo o negativo. El resultado será del mismo tipo que la base, y por tanto en algunos casos conviene utilizar la función Exp, para evitar un desborde del sistema.

ROUND(m,n).- Produce un redondeo de m al número de decimales indicado por n. Consultar la ayuda, pues cada gestor tiene un funcionamiento distinto, y en SQL SERVER, se puede sustituir con la función de conversión, que ya veremos más adelante.

SIGN(n).- Devuelve -1 si n<0, 0 si n=0 y 1 si n>0.

SQRT(n).- Realiza la raíz cuadrada de n. Si n < 0 devuelve NULL.

Todas las funciones trigonométricas, que toman su argumento en radianes.

FUNCIONES DE CADENA DE CARACTERES

ASCII(s).- Devuelve el valor ASCII de la cadena s.

CHAR(n).- Admite un argumento entre 0 y 255, en otro caso devuelve el valor NULL. Devuelve el carácter con código ASCII n, y se utiliza para introducir caracteres de control utilizados con frecuencia.

CHARINDEX(s1,s2[,n]).- Busca la cadena s1 en la cadena (generalmente columna s2), desde la posición indicada en n. Si $n \leq 0$, comienza desde la posición 1.
En caso de que la cadena buscada no se encuentre, devuelve el valor 0, en otro caso, devuelve la posición de la primera vez que aparece dicha cadena.

LEFT(s,n) Y RIGHT(s,n).- devuelve n caracteres por la izquierda o derecha respectivamente.

LEN(s).- Devuelve el número de caracteres de s eliminando blancos a derecha de la cadena.

LOWER(s) Y POWER(s).- Convierten respectivamente a minúscula o mayúscula la cadena s.

LTRIM(s) y RTRIM(s).- eliminan los espacios por la izquierda o derecha respectivamente.

QUOTENAME(s).- Se utiliza para crear identificadores válidos en SQL SERVER.

REPLACE(s1,s2,s3).- Sustituye la cadena s2 en s1 por la cadena s3.

REPLICATE(s,n).- Repite la cadena s tantas veces como lo indique n. En caso que n sea negativo, devuelve NULL.

REVERSE(s).- Devuelve la cadena s al revés.

DIFFERENCE Y SOUNDEX.- ambas toman como argumento dos cadenas de caracteres realizando la comparación de las mismas. Devuelven un valor entre 0 y 4.

SPACE(n).- Devuelve tantos espacios como indique n.

FUNCIONES DE CONVERSIÓN. (CAST Y CONVERT)

Convierte un tipo de datos a otro, la sintaxis es igual en ambos casos:

CAST (campo AS tipo)

FUNCIONES DE FECHA Y HORA.

En todas las funciones de fecha:

Parte de la fecha	Abreviaturas
Year	**yy, yyyy**
quarter	**qq, q**
Month	**mm, m**
dayofyear	**dy, y**
Day	**dd, d**
Week	**wk, ww**
weekday	**dw, w**
Tour	**hh**
minute	**mi, n**
second	**ss, s**
millisecond	**ms**

DATEADD(parte de fecha, numero, fecha).- Obtiene una nueva fecha, después de sumar el numero a la parte indicada.

DATEDIFF(parte de fecha, fecha inicio, fecha fin).- indica la diferencia entre las dos fechas en el valor indicado en parte de fecha.

DATENAME(parte de fecha, fecha).- devuelve una cadena de caracteres extrayendo la parte especificada en parte de fecha.

DATEPART(parte de fecha, fecha).- Devuelve un valor numérico con la parte de la fecha indicada.

DAY(fecha).- devuelve el día del mes. Equivalente a datepart.

GETDATE().- devuelve la fecha y la hora del sistema.

MONTH(fecha).- Devuelve el mes del año en número. Equivalente a datepart.

YEAR(fecha).- Devuelve el año de la fecha. Equivalente a datepart.

OTRAS FUNCIONES:

FUNCION ISNULL.- (EN OTROS GESTORES PUEDE SER NVL).

ISNULL(x,expresión).- Devuelve x si no es nula, y expresión en el caso de que lo fuera. Hay que tener en cuenta, que cualquier valor operado con NULL, será NULL, por tanto es una función muy utilizada.

Ejemplo: Obtener el salario que cobrarán los empleados este mes. Estos cobrarán el salario más la comisión, pues si no utilizamos la función isnull, todos aquellos empleados que carecen de comisión este mes no cobrarían.

select apellido,salario+comision salario
from emp2

APELLIDO	SALARIO
SANCHEZ	NULL
ARROYO	1484.0000
SALA	1368.0000
JIMENEZ	NULL
MARTIN	1086.0000
NEGRO	NULL
CEREZO	NULL
GIL	NULL
REY	NULL
TOVAR	1172.0000
ALONSO	NULL
JIMENO	NULL
FERNANDEZ	NULL
MUÑOZ	NULL

Se puede ver que quien no tiene comisión no cobra

La solución es sumarle al salario el neutro de la suma si no tienen comisión para evitar lo anterior:

select apellido,salario+isnull(comision,0) salario
from emp2

APELLIDO	SALARIO
SANCHEZ	625.0000
ARROYO	1484.0000
SALA	1368.0000
JIMENEZ	2324.0000
MARTIN	1086.0000
NEGRO	2227.0000
CEREZO	1914.0000
GIL	2344.0000
REY	3907.0000
TOVAR	1172.0000
ALONSO	859.0000
JIMENO	742.0000
FERNANDEZ	2344.0000
MUÑOZ	1016.0000

FUNCION CASE (EN OTROS GESTORES DECODE).-

IGUALDAD.-

```
Nombre_columna o alias =
CASE nombre_columna
        WHEN valor1 THEN cod1
        ...
        ELSE <valor por defecto>
END
```

TAMBIÉN:

```
CASE nombre_columna
        WHEN valor1 THEN cod1
        ...
        ELSE <valor por defecto>
END alias o nombre_columna
```

Ejemplo: Obtener, de los enfermos ingresados, su apellido, sala y teniendo en cuenta esta, si está en recuperación, aparecerá mejorando, si cuidados intensivos aparecerá grave, si Psiquiátrico entonces caso mental, si cardiología entonces caso del corazón y en otro caso, paciente en estudio:

```
select apellido, nombre sala, case nombre
            when 'recuperacion' then 'Mejorando'
            when 'Cuidados intensivos' then 'Grave'
            when 'psiquiatrico' then 'Caso Mental'
            when 'cardiologia' then 'Caso del Corazon'
            else 'Paciente en estudio'
            end comentario

from ocupacion o, enfermo e, sala s
where  o.inscripcion=e.inscripcion and
       o.sala_cod=s.sala_cod and
       o.hospital_cod=s.hospital_cod
```

la salida es:

APELLIDO	SALA	COMENTARIO
Laguia M.	Cuidados Intensivos	Grave
Fernandez M.	Cuidados Intensivos	Grave
Neal R.	Cardiologia	Caso del Corazon
Cervantes M.	Psiquiatrico	Caso Mental
Miller G.	Psiquiatrico	Caso Mental
Fraser A.	Maternidad	Paciente en estudio
Domin S.	Cardiologia	Caso del Corazon
Ruiz P.	Psiquiatrico	Caso Mental

FUNCIÓN CASE CON VALORES DE COMPARACIÓN

```
'Nombre_columna o alias' =
        CASE nombre_columna
                WHEN expresión booleana THEN cod1
                ...
                ELSE <valor por defecto>
        END
```

Obtener los empleados, su comisión, y el mensaje mas de 1000 euros, menos de 1000 euros, o el mensaje no tiene comisión

```
select apellido,comision, 'mensaje'=
case
when comision is null then 'no tiene comision'
when comision >=1000 then 'mas de mil euros'
     else 'menos de mil' end
from emp2
```

APELLIDO	COMISION	COMENTARIO
SANCHEZ	NULL	no tiene comisión
ARROYO	234.00	menos de mil
SALA	.0000	menos de mil
JIMENEZ	NULL	no tiene comisión
MARTIN	109.00	menos de mil
NEGRO	NULL	no tiene comisión
CEREZO	NULL	no tiene comisión
GIL	NULL	no tiene comisión
REY	NULL	no tiene comisión
TOVAR	0.00	menos de mil
ALONSO	NULL	no tiene comisión
JIMENO	NULL	no tiene comisión
FERNANDEZ	NULL	no tiene comisión
MUÑOZ	NULL	no tiene comisión

FUNCIONES DE CONJUNTO

LAS MAS UTILIZADAS SON:

- **AVERAGE AVG**
- **COUNT DISTINCT**
- **COUNT(*)**
- **MAX**
- **MIN**
- **SUM**

Todas ellas se pueden utilizar junto con operadores de comparación y con expresiones de valores, pero no pueden ser mezcladas en una consulta a menos que exista una cláusula GROUP BY.

Ej.

```
SELECT funcion, AVG(salario)          ES INCORRECTA
FROM emp2
```

```
SELECT funcion, AVG(SALARIO)          ES CORRECTA
FROM emp2
GROUP BY funcion
```

EN TODAS O CASI TODAS ELLAS LOS VALORES NULL SE IGNORAN.

***FUNCION AVERAGE (AVG)**

Calcula la media aritmética de los valores indicados en la columna. Su sintaxis es:

AVG([DISTINCT] nombre_columna)

- DISTINCT elimina los valores duplicados para el cálculo de la media.
- No se puede aplicar a cadenas de caracteres.

Ejemplos:

1)
```
Select AVG(salario)
From emp2
```
Obtiene la media salarial de todos los empleados.

2)
```
Select AVG(salario)
From emp2
Where funcion=’analista’
```
Obtiene la media salarial de todos los analistas

3)
```
Select AVG(salario)
From emp2
Group by funcion
```
Obtiene la media salarial de los empleados agrupados por la función que desempeñan.

*** FUNCIÓN COUNT**

Cuenta el número de valores distintos de NULL que hay en una columna.

Su sintaxis es:

COUNT ({ [DISTINCT] nombre_columna}{*})

- La opción distinct, cuenta los valores distintos de la columna y como puede verse es opcional, si no se indica se cuentan todos los valores de la columna distintos de null.
- La opción * cuenta filas completas. (se supone que no habrá filas iguales, por considerar que la base de datos tiene la redundancia eliminada).

***FUNCIONES MAX Y MIN**

Devuelve los valores máximo y mínimo respectivamente de la columna indicada.

MAX(nombre_columna)
MIN(nombre_columna)

No es aplicable a distinct, pues no tendría sentido alguno.

Ej.
Obtener el nombre y el peso de la parte que más pesa.

```
SELECT NOMP
FROM P
WHERE PESO= (SELECT MAX(PESO)
                    FROM P)
```

***FUNCIÓN SUM**

Suma los valores de la columna especificada, su argumento será numérico y el tipo de salida igual al del argumento. La sintaxis es:

SUM([DISTINCT] nombre_columna)

Elimina como siempre los nulos y distinct también tiene el mismo significado.

Ej.
Obtener la suma salarial de todos los empleados:

```
Select sum(salario)
From emp2
```

SELECCIÓN DE CONJUNTOS

Se utiliza para calcular propiedades de uno o más conjunto de filas.

Si es más de un conjunto de filas el seleccionado, la cláusula GROUP BY controla que las filas de la tabla original sean agrupadas en una tabla temporal. La cláusula HAVING se emplea para controlar cuál de esos grupos se desea visualizar.

Por tanto, la consideración de las cláusulas en tiempo de ejecución es:

1.- FROM	Obtiene las tablas en memoria
2.- WHERE	Filtra las filas que cumplen la condición
3.- GROUP BY	Crea una tabla grupo nueva
4.- HAVING	Filtra los grupos
5.- SELECT	Muestra el nombre de grupo o funciones de grupo
6.- ORDER BY	Ordena la salida

*CLÁUSULA "GROUP BY"

Realiza la función de agrupación en el resultado, para distintos valores de las columnas especificadas en la cláusula, creando una tabla temporal con el resultado de la agrupación.

SINTAXIS:

```
SELECT clumna_agrupación1,...,columna_agrupacionN, funcion_grupo
FROM nombre_tabla
[WHERE condicion(es) de fila]
GROUP BY clumna_agrupación1,...,columna_agrupacionN
```

Las columnas que aparezcan en la select, deben aparecer forzosamente en el group by y no al contrario, es decir puede agruparse por columnas que después no se muestren en la select.

Ej.

Obtener la suma salarial de todos los empleados:

```
SELECT SUM(SALARIO) SUMA_SALARIAL FROM EMP2

SUMA_SALARIAL
22678.00
```

Obtener la suma salarial de los empleados por departamento

```
SELECT DEPT_NO, SUM(SALARIO) SUMA_SALARIAL
FROM EMP2
GROUP BY DEPT_NO
```

DEPT_NO	SUMA_SALARIAL
10	6837.00
20	8496.00
30	7345.00

***CLÁUSULA "HAVING"**

Especifica una restricción en la tabla agrupada que resulta de la cláusula group by, eliminando aquellos grupos que no satisfagan la condición especificada.

El uso de **having** requiere forzosamente que se haya especificado la cláusula **group by**, y los campos que aparezcan deben haber sido objeto de grupo o también puede utilizarse una función de grupo.

*En la where **NUNCA** puede ir una función de grupo.*

La where impone restricciones a las filas y having hace lo mismo con los grupos.

SINTAXIS

```
SELECT [columna(s)_agrupadas], funcion_grupo
FROM tabla(s)
WHERE condicion(es)
GROUP BY columna(s) de agrupación
HAVING propiedad específica de grupo.
```

"propiedad específica de grupo", se construirá con predicados y conectores lógicos igual que la where.

Ej.

1. Obtener el salario medio anual de todos los puestos de trabajo en los que hay más de un empleado.

```
SELECT OFICIO,COUNT(*)n_empleados,avg(salario)MEDIA
FROM EMP2
GROUP BY OFICIO
HAVING count(*)>1
```

La salida es:

OFICIO	N_EMPLEADOS	MEDIA
ANALISTA	2	2344.00
DIRECTOR	3	2155.00
EMPLEADO	4	810.50
VENDEDOR	4	1094.00

2. Obtener la suma de los empleados por departamento, teniendo en cuenta solamente aquellos cuyo salario sea superior a 300€.

```
select dept_no, sum(salario)suma_salarial
from emp2
where salario>300
group by dept_no
```

Dep._no	suma_salarial
10	6837.0000
20	8496.0000
30	7345.0000

3. Obtener la suma de los salarios de cada departamento cuya suma sea mayor a 600€.

```
select dept_no, sum(salario)suma_salaria
from emp2
group by dept_no
having sum(salario)>600
```

Dep._no	suma_salarial
10	6837.0000
20	8496.0000
30	7345.0000

OPERADORES EXISTS Y NOT EXISTS

(Utilizaremos en el ejemplo las tablas de s,p,j,spj)

***EXISTS**

Se utiliza en una subconsulta para comprobar la existencia de filas devuelta en la misma. En muchos casos se puede sustituir por el operador IN, aunque habrá ocasiones en las cuales, como siempre, tenga que ser de uso obligatorio.

Ej. Obtener los nombres de los proveedores que venden la parte P2.

```
select noms
from s
where exists (select *
              from spj
              where s.s#=spj.s# and p#='p2')
```

la select de la subconsulta, siempre nunca recupera un campo concreto, siempre son filas enteras, y el tipo de la subconsulta, siempre es una referencia cruzada.

Recuperará el nombre de los proveedores de la tabla s, siempre que haya alguna fila seleccionada en la tabla spj, del proveedor en estudio que venda la parte p2.

***NOT EXISTS**

Se realiza la recuperación de la fila seleccionada en la select, siempre que el resultado de la ejecución de la subselect, sea conjunto vacío, es decir, no haya filas seleccionadas.

Este operador es imposible sustituirlo, siendo en algunas consultas de usos absolutamente necesario.

Ej. 1.- Obtener el nombre de aquellos proveedores que no vende la parte p2. La consulta sería pensada así, seleccionamos el nombre del proveedor de la tabla s, siempre que no haya en spj una venta suya de la parte p2.

```
select noms
from s
where not exists (select *
                  from spj
                  where s#=s.s# and p#='p2')
```

en este caso se puede sustituir por not in.

Ej. 2.- Obtener el nombre de aquellos proveedores que venden todas las partes. El planteamiento es buscar aquellos poveedores cuando no exista una parte que no venda. En este caso el operador not exists es insustituible.(*)

```
select noms
from s
where not exists (select *
                  from p
                  where not exists (select *
                                    from spj
                                    where s#=s.s# and p#=p.p#))
```

NOMS

Aldana

TRANSACCIONES

Son sentencias transaccionales aquellas que someten a modificación el contenido de la base de datos.

EN LINEAS GENERALES

Generalmente, las sentencias transaccionales, son aquellas que modifican el contenido de las tablas de la base de datos, es decir, las sentencias INSERT, UPDATE y DELETE (inserción de fila en tabla, modificación de las filas de tabla y borrado de filas en tabla).

La transacción comienza: al inicio de la sesión, al finalizar la sesión o al finalizar la sesión anterior.

La sesión finaliza explícitamente con la sentencia COMMIT (aceptación de los cambios), o la sentencia ROLLBACK (desestimar los cambios)

Se realiza un COMMIT implícito, al finalizar la sesión o al ejecutar una sentencia de LDD (CREATE, DROP o ALTER).

Se realiza un ROLLBACK implícito si se produjera una caída del sistema.

TRANSACCIONES EN SQL SERVER 2000

Son sentencias transaccionales, todas aquellas que modifican tanto el contenido de las tablas de la base de datos, como los objetos de la base de datos. (INSERT, UPDATE, DELETE, CREATE, DROP y ALTER).

La transacción se inicia con la sentencia BEGIN TRNS [nombre] y termina con COMMIT [nombre] o ROLLBACK [nombre].

Si no se inicia la transacción de forma explícita, esta se acepta, porque el sistema no permite aceptar ni rechazar de forma explícita. Una caída del sistema supone el rechazo de la transacción.

Mientras se está realizando una sentencia transaccional en una tabla se produce un bloqueo a nivel de fila, para evitar una transacción simultanea desde otro lugar y que se produzca un problema de inconsistencia en la base de datos. (recordemos que la base de datos es integrada, compartida o distribuida)

SENTENCIAS TRANSACCIONALES: INSERT, UPDATE Y DELETE.

INSERT.-

Añade filas nuevas en una tabla.

Sintaxis:

INSERT [INTO] nombre_tabla [(nombre_columna1...nombre_columnaN)]
VALUES (valor1...valorN)

La lista de valores, será en la misma secuencia que corresponda al orden en que se enumeraron las columnas. En caso de omitirse el nombre de estas, habrá que introducir el valor de los campos en el mismo orden en que se describieron en la creación de la tabla, sin omitir ninguno.

- Cuando un campo tenga valor NULL, se pondrá exactamente dicho valor es decir NULL o se omitirá el campo al introducir los nombres de la tabla.
- Si el valor que toma una columna, es el especificado por defecto en la creación de la tabla, el término a poner será DEFAULT.
- Si una columna, se le dio la condición de itentity (valor autonumérico), esta columna se omite en el insert.
- Las fechas y valores alfanuméricos, van entre comilla simple.
- Los tipos, las longitudes y las restricciones de las columnas deben respetarse, en caso contrario el gestor dará un mensaje de error.
- Los valores pueden provenir de una consulta

IMPORTANTE.- Hay que tener cuidado en la asignación del valor de los datos, pues, si estos son compatibles con la columna correspondiente, el valor introducido será incorrecto y el gestor no avisará.

UPDATE.-

Se utiliza para actualizar el contenido de una o varias columnas, en una, varia o todas las filas de una tabla (por supuesto, se deben respetar las restricciones asignadas a las columnas en la creación de la tabla).

Sintaxis:

```
UPDATE nombre_tabla
SET columna1=valor_nuevo
      [,columnaN=valor nuevo]
[WHERE condicion]
```

La sentencia where puede ser una subconsulta, de la misma forma que se utiliza en la select.

DELETE.-

Elimina filas completas de una tabla, pero aunque borremos todo el contenido de la tabla y esta quede vacía, el objeto, es decir la tabla seguirá existiendo en el catálogo.

Puede ser, que el gestor no permita el borrado de alguna fila, porque esto sea motivo de que se incumpla alguna restricción de integridad referencial en la base de datos, avisando del suceso.

Sintaxis:

```
DELETE
FROM nombre_tabla
[WHERE condición]
```

La cláusula where, igual que siempre.