**MANUAL BASES DE DATOS SQL SERVER**

**LENGUAJE ANSI SQL**

* Las bases de datos se pueden diferenciar en dos grandes familias:
  + **Bases de datos de Usuario:** Bases de datos cuya administración se basa en un fichero, su seguridad es muy baja y no admite múltiples conexiones.
  + **Bases de datos empresariales:** Son bases de datos de servidor, admiten múltiples conexiones simultáneas y el nivel de seguridad y administración es muy alto.

Existen 3 tipos de instrucciones para el lenguaje en SQL.

* **Lenguaje de control de datos (DDL):** Creación y eliminación de tipos de datos y objetos.

. CREATE Crear Objeto

. ALTER Modificar los datos creados

. DROP Eliminar el Objeto

* **Lenguaje de control de datos (DCL):** Se basa en los derechos que tiene el usuario sobre la base de datos (Permisos).

. GRANT Dar permisos a un usuario para efectuar determinadas

Instrucciones

. DENY Eliminar el permiso que se ha concedido con el GRANT

. REVOKE Eliminar todos los permisos

* **Lenguaje de manipulación de datos (DML):** Desarrollo de la programación de la base de datos.

. SELECT

. INSERT

. UPDATE

. DELETE

# Elementos de sintaxis:

Directivas de procesos por lotes

* **GO:** Envía lotes de instrucciones de TRANSACT SQL a las herramientas y utilidades (Sirve para separar bloques de instrucciones)
* **EXEC O EXECUTE:** Ejecuta funciones definidas por el usuario, procedimientos de sistema y procedimientos almacenados.

**Comentarios en SQL:**

* En línea: **--**
* En Bloque: **/\*** *comentario* **\*/**
* **Identificadores para los objetos:**

Los nombres que se le dan a las tablas, lo primero es que no pueden empezar por un número, deben empezar por un signo alfabético, pueden incluir el guion bajo ( \_), la arroba @ y la almohadilla #.

**Tipos de datos más comunes:**

* **Numéricos:** 
  + - Enteros à int, tinyint, smallint, bigint
    - Decimales à numeric, decimal, money, smallmoney
    - Coma Flotante à float, real
* **Fechas:** 
  + - datetime à Fecha y Hora
    - smalldatetime à Fecha
* **Caracteres:** 
  + - Ancho fijo: char, varchar
    - Ancho Variable: nchar, nvarchar
* **Texto e Imagen:**
  + - Varchar(Max)
    - Ntext
    - Rowversion
* **Binario:** 
  + - Binary, varbinary à Valores tipo byte
    - Bit à Un solo bit (1 o ninguno)
* **Identificadores Unicos:**

Representa un identificador global único (GUID)

Si queremos que no se repita el dato en la base de datos, usamos este identificador

* + - Uniqueidentifier

**CREACIÓN DE TABLAS POR CÓDIGO**

* Debemos entrar en el Analizador de consultas desde la barra de herramientas. (**Nueva Consulta)**
* **Creación, eliminación y modificación de una tabla.**

**Sintaxis:**

**Creación de una tabla:**

CREATE TABLE NombreTabla (NombreColumna TipoDato [NULL | NOT NULL])

* Si no ponemos nada, el valor por defecto para la creación de una tabla es NULL.
* Escribir Null o Not null implica una restricción, ya que estamos obligando al usuario a escribir el dato.

**Eliminación una tabla:**

DROP TABLE NombreTabla

**Modificar una tabla: (Agregar, modificar y eliminar columnas)**

ALTER TABLE NombreTabla ADD NombreColumna TipoDato [NULL | NOT NULL ]

El valor por defecto (NULL) no se pone porque no lo coge en este momento.

ALTER TABLE emp ADD Edad INT NOT NULL

Modificar una columna

ALTER TABLE emp ALTER COLUMN Edad INT

Para eliminar una columna

ALTER TABLE NombreTabla DROP COLUMN NombreColumna

------------------------------ Mandatos entrados ------------------------------

CREATE TABLE PRUEBA2

(IDPRUEBA2 INTEGER NOT NULL

, PRUEBA VARCHAR(30));

------------------------------------------------------------------------------

**La instrucción ha finalizado correctamente.**

**RESTRICCIONES DE LAS TABLAS**

**Las restricciones son “reglas” que incluimos en una base de datos para mantener la integridad referencial de las tablas. Por ejemplo, si tenemos una columna para incluir la fecha de nacimiento de una persona, no deberiamos permitir al usuario poder incluir fechas posteriores al día de hoy.**

# Tipos de restricciones

|  |  |
| --- | --- |
| RESTRICCION | ACCION |
| **NULL / NOT NULL** | **Obliga al usuario a insertar datos en columnas** |
| **PRIMARY KEY** | **No puede admitir nulos.**  **Solamente puede haber una por tabla.**  **Impide datos repetidos en la columna.**  **Crea indices para poder realizar búsquedas en la tabla de forma más óptima.**  **Puede ser de una o varias columnas.** |
| **UNIQUE** | **No puede admitir nulos.**  **Puede haber más de una por tabla.**  **Impide datos repetidos en la columna.**  **Crea indices para poder realizar búsquedas en la tabla de forma más óptima.**  **Puede ser de una o varias columnas.** |
| **FOREIGN KEY** | **Crea una relación entre dos tablas.**  **Impide incluir datos que no estén relacionados entre las dos tablas.**  **Debe hacer referencia a un campo Primary Key o Unique.** |
| CHECK | **Impide la inserción de datos que no correspondan a una regla establecida por el usuario.** |

**Sintaxis de las restricciones:**

* **NOT NULL:**

**CREATE TABLE TABLA**

**(CAMPO1 INT NOT NULL, CAMPO2 VARCHAR(30))**

# PRIMARY KEY

**ALTER TABLE NOMBRETABLA**

**ADD CONSTRAINT PK\_NOMBRE\_RESTRICCION**

**PRIMARY KEY (CAMPO)**

Nota: Puede hacer referencia a más de un campo.

**ALTER TABLE NOMBRETABLA**

**ADD CONSTRAINT PK\_NOMBRE\_RESTRICCION**

**PRIMARY KEY (CAMPO1, CAMPO2)**

# UNIQUE

**ALTER TABLE NOMBRETABLA**

**ADD CONSTRAINT U\_NOMBRE\_RESTRICCION**

**UNIQUE (CAMPO)**

Nota: Puede hacer referencia a más de un campo.

**ALTER TABLE NOMBRETABLA**

**ADD CONSTRAINT U\_NOMBRE\_RESTRICCION**

**UNIQUE (CAMPO1, CAMPO2)**

# FOREIGN KEY

**Una clave foranea, hace referencia a un dato que se encuentra en una tabla principal. Por ejemplo, si tengo una tabla de Cliente y Pedidos, un pedido siempre debe hacer referencia a un Cliente. La tabla principal serían los clientes y la tabla secundaria serian los Pedidos.**

**ALTER TABLE NOMBRETABLASECUNDARIA**

**ADD CONSTRAINT FK\_NOMBRE\_RESTRICCION**

**FOREIGN KEY (CAMPOREFERENCIA)**

**REFERENCES NOMBRETABLAPRINCIPAL (CAMPOPRINCIPAL)**

**Existe la posibilidad de decidir que queremos hacer en el caso de que se eliminen o modifiquen registros que hacen referencia, por ejemplo si eliminamos un cliente que tiene pedidos asociados, podemos realizar varias acciones:**

* **Impedir la eliminación (Por defecto): Si intentamos eliminar un cliente con pedidos, la Foreign Key nos impide ésta acción.**

**ALTER TABLE NOMBRETABLASECUNDARIA**

**ADD CONSTRAINT FK\_NOMBRE\_RESTRICCION**

**FOREIGN KEY (CAMPOREFERENCIA)**

**REFERENCES NOMBRETABLAPRINCIPAL (CAMPOPRINCIPAL)**

**ON DELETE RESTRICT**

**ON UPDATE RESTRICT;**

* **Eliminación en cascada. Si intentamos eliminar un cliente con pedidos, se eliminará el cliente y sus pedidos asociados.**

**ALTER TABLE NOMBRETABLASECUNDARIA**

**ADD CONSTRAINT FK\_NOMBRE\_RESTRICCION**

**FOREIGN KEY (CAMPOREFERENCIA)**

**REFERENCES NOMBRETABLAPRINCIPAL (CAMPOPRINCIPAL)**

**ON DELETE CASCADE**

**ON UPDATE CASCADE;**

* **Establecer nulos. Si intentamos eliminar un cliente con pedidos, se eliminará el cliente y sus pedidos quedarán con el campo de referencia a null.**

**ALTER TABLE NOMBRETABLASECUNDARIA**

**ADD CONSTRAINT FK\_NOMBRE\_RESTRICCION**

**FOREIGN KEY (CAMPOREFERENCIA)**

**REFERENCES NOMBRETABLAPRINCIPAL (CAMPOPRINCIPAL)**

**ON DELETE SET NULL**

**ON UPDATE SET NULL;**

# CHECK

**ALTER TABLE NOMBRETABLA**

**ADD CONSTRAINT CK\_NOMBRE\_RESTRICCION**

**CHECK** **(CAMPO = CONDICION)**

**Ejemplo: Impedir que un campo entero tenga valores negativos.**

**ALTER TABLE NOMBRETABLA**

**ADD CONSTRAINT CK\_NEGATIVOS**

**CHECK** **(NUMERO >= 0)**

**Consulta para recuperar todas las restricciones de una tabla:**

SELECT \* FROM SYSIBM.SYSTABCONST;

# CONSULTAS DE SELECCIÓN

Para realizar consultas de selección sobre los registros de la table se utiliza la instrucción SELECT.

Sintaxis:

*SELECT CAMPO1, CAMPO2, CAMPO3 FROM TABLA*

Devolvería los datos del Campo1, Campo2 y Campo3

También sería válida la siguiente sentencia:

*SELECT \* FROM TABLA*

Devolvería todos los datos de la tabla.

La utilización del asterisco para las consultas es algo muy extendido, ya que es más sencillo de utilizar como programador, pero a nivel de base de datos, es más costoso para el motor de la base de datos.

Recomendación: Deberíamos utilizar siempre el nombre de los campos.

Ejemplo:

SELECT APELLIDO, OFICIO, SALARIO FROM EMP

SELECT \* FROM EMP

**ORDER BY**

* + - Ordena las columnas por el campo especificado a continuación del Order By.
    - Podemos ordenar de forma ascendente (ASC) o descendente (desc).
    - Si no escribimos nada después del orden de un campo, por defecto siempre será ASC.

Sintaxis:

Select Apellido, Oficio, Salario, Dept\_no from emp

ORDER BY APELLIDO DESC

Select Apellido, Oficio, Salario, Dept\_no from emp

ORDER BY APELLIDO ASC

Podemos ordenar la consulta de resultados por más de un campo:

Select Apellido, Oficio, Salario, Dept\_no from emp

ORDER BY OFICIO, SALARIO DESC

Estaríamos ordenando todos los empleados por su oficio, y dentro de éste orden, por su salario de menor a mayor.

**WHERE**

* La instrucción where de una consulta, filtra los resultados de la consulta por los resultados que escribamos a continuación del WHERE.

Sintaxis:

SELECT CAMPO1, CAMPO2 FROM TABLA WHERE CAMPO1=VALOR

Operadores de Comparación:

|  |  |
| --- | --- |
| OPERADOR | VALOR |
| > | Mayor |
| < | Menor |
| >= | Mayor o Igual |
| <= | Menor o Igual |
| = | Igual |
| <> | Diferente |

**Nota: DB2 diferencia es CaseSensitive. Diferencia entre mayúsculas y minúsculas en los datos, es decir, no es lo mismo el dato VENDEDOR que vendedor.**

**Todos los datos que no sean numéricos, deben ser representados con comillas simples en la instrucción where. (Fechas y Varchar)**

Ejemplos:

* Mostrar el apellido, el oficio y el salario de los empleados cuyo salario sea mayor a 200.000

SELECT APELLIDO, OFICIO, SALARIO FROM EMP

WHERE SALARIO > 200000

* Mostrar el apellido, el oficio y el salario de los empleados cuyo oficio sea Director

SELECT APELLIDO, OFICIO, SALARIO FROM EMP

WHERE OFICIO=’DIRECTOR’

No tienen nada que ver los datos que mostramos en SELECT con el filtro de la instrucción WHERE. Podemos mostrar unos datos en la consulta y filtrar por otra columna diferente de la tabla que no está dentro de la consulta:

SELECT APELLIDO, FUNCION FROM PLANTILLA

WHERE TURNO=’T’

Podemos combinar perfectamente WHERE con ORDER BY. **La última instrucción de cualquier consulta de selección será siempre ORDER BY (opcional).**

Ejemplo:

* Mostrar el apellido y la función de la plantilla del turno de tarde ordenados por su salario de forma ascendente.

SELECT APELLIDO, FUNCION FROM PLANTILLA

WHERE TURNO=’T’ ORDER BY SALARIO

Operadores Relacionales:

|  |  |
| --- | --- |
| OPERADOR | VALOR |
| AND | Y |
| OR | O |
| NOT | NEGACION |
| BETWEEN | ENTRE |

* Los operadores relacionales permiten filtrar datos por más de un campo, dependiendo de las condiciones de la consulta o los datos que necesitemos mostrar.

Ejemplo:

* Mostrar todos los datos de la plantilla que sean enfermeras y trabajen en el turno de noche.

SELECT \* FROM PLANTILLA

WHERE FUNCION=’ENFERMERA’ AND TURNO=’N’

**Mostrará todos los registros que cumplan las dos condiciones, que sean enfermeras y que trabajen en el turno de noche.**

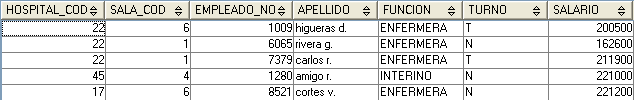


* Mostrar todos los datos de la plantilla que sean enfermeras o trabajen en el turno de noche.

SELECT \* FROM PLANTILLA

WHERE FUNCION=’ENFERMERA’ AND TURNO=’N’

**Mostrará todos los registros que trabajen en el turno de noche independientemente de su función y las enfermeras, independientemente de su turno.**



* Mostrar todos los doctores que no trabajen en el hospital 45.

SELECT \* FROM DOCTOR WHERE NOT HOSPITAL=45

* Siempre que hagamos una negación, debemos intentar hacer la consulta en forma positiva, ya que el NOT hace que las consultas vayan más lentas y perderemos eficacia (VELOCIDAD) en el momento de mostrar los datos:

**SELECT \* FROM DOCTOR WHERE HOSPITAL <> 45**

* **Between** busca el valor comprendido entre un rango de datos.

Mostrar todos los doctores del hospital cuyo salario esté comprendido entre 250.000 y 300.000

SELECT \* FROM DOCTOR

WHERE SALARIO BETWEEN 250000 AND 300000;



Mostrar todos los doctores del hospital cuyo salario esté comprendido entre 250.000 y 300.000 y su especialidad sea Pediatria.

SELECT \* FROM DOCTOR

WHERE SALARIO BETWEEN 250000 AND 300000

AND ESPECIALIDAD=’Pediatria’;



**Campos Calculados**

Los campos calculados son campos que no están en la tabla, pero que se muestran a partir de un cálculo de otros campos de la tabla.

Supongamos que tenemos la siguiente tabla PERFUMES:

|  |  |  |
| --- | --- | --- |
| PRODUCTO | PRECIO | CANTIDAD |
| Hugo Boss Bottle | 55 | 2 |
| Carolina Herrera 212 | 51 | 1 |
| UltraViolet Paco Rabanne | 62 | 3 |

Si quiero mostrar los datos de la tabla, bastaría con escribir lo siguiente:

SELECT PRODUCTO, PRECIO, CANTIDAD FROM PERFUMES;

Pero si lo que necesito es averiguar el precio total facturado, tendré que multiplicar el Precio por la Cantidad, y es un dato que no existe en la tabla, pero que puedo recuperar a partir de un cálculo de otros datos de la tabla. Eso es un campo calculado.

SELECT PRODUCTO, PRECIO, CANTIDAD, PRECIO \* CANTIDAD

FROM PERFUMES;

|  |  |  |  |
| --- | --- | --- | --- |
| PRODUCTO | PRECIO | CANTIDAD |  |
| Hugo Boss Bottle | 55 | 2 | **110** |
| Carolina Herrera 212 | 51 | 1 | **102** |
| UltraViolet Paco Rabanne | 62 | 3 | **124** |

Como podemos comprobar, nos devuelve el resultado esperado, pero siempre que tengamos un campo calculado, debemos de darle un nombre a la columna de SALIDA para poder recuperar y visualizar correctamente los datos. Para ello se utiliza la palabra clave AS a continuación de la consulta, escribiendo el alias de la columna a continuación:

SELECT PRODUCTO, PRECIO, CANTIDAD

, **PRECIO \* CANTIDAD AS “TOTAL”**

FROM PERFUMES;

|  |  |  |  |
| --- | --- | --- | --- |
| PRODUCTO | PRECIO | CANTIDAD | **TOTAL** |
| Hugo Boss Bottle | 55 | 2 | **110** |
| Carolina Herrera 212 | 51 | 1 | **102** |
| UltraViolet Paco Rabanne | 62 | 3 | **124** |

Podemos ordenar por el “ALIAS” de los campos calculados

SELECT PRODUCTO, PRECIO, CANTIDAD

, PRECIO \* CANTIDAD AS “TOTAL”

FROM PERFUMES

**ORDER BY TOTAL**;

**Operador IN**

El operador IN se utiliza para consultas sobre un mismo campo en las que tengamos que utilizar múltiples valores para el filtro de los datos.

Supongamos que necesito saber todos los empleados de la plantilla que trabajan en el hospital 45.

SELECT \* FROM PLANTILLA WHERE HOSPITAL\_COD=45

Ahora quiero también los que trabajen en el hospital 22.

SELECT \* FROM PLANTILLA WHERE HOSPITAL\_COD=45

OR HOSPITAL\_COD=22

Ahora quiero también los que trabajen en el hospital 19.

SELECT \* FROM PLANTILLA WHERE HOSPITAL\_COD=45

OR HOSPITAL\_COD=22 OR HOSPITAL=19

La lista de posibilidades para el campo Hospital\_cod puede ser interminable. Si yo ahora quisiera también los del hospital 14, 90, y 60, tendríamos que hacer una consulta enorme para varios valores del Hospital.

El operador IN, permite realizar la consulta sobre un mismo campo y ofreciéndole múltiples valores para ese campo.

Ejemplo:

Quiero todos los empleados de la plantilla que trabajen en el hospital 45, 22, 19, 90, 14 y 60.

**SELECT \* FROM PLANTILLA**

**WHERE HOSPITAL\_COD IN (45, 22, 19, 90, 14, 60)**

Como se puede comprobar los resultados de la consulta son los mismo y ahorramos mucho código.

También podemos utilizar el operador **NOT IN**, para preguntar lo contrario.

Ejemplo:

Quiero todos los empleados de la plantilla que **NO** trabajen en el hospital 45, 22, 19, 90, 14 y 60.

SELECT \* FROM PLANTILLA

WHERE HOSPITAL\_COD **NOT** IN (45, 22, 19, 90, 14, 60)

**Operador DISTINCT**

El operador Distinct devuelve los datos NO REPETIDOS de un campo determinado de la tabla, es decir, oculta los datos repetidos y solamente los muestra una vez.

Miremos los datos de la tabla empleados:



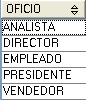
Si necesitase saber los oficios de los empleados, me mostraría todos los oficios, pero con elementos repetidos:

SELECT OFICIO FROM EMP;



La cláusula DISTINCT ofrece los mismo resultados pero quita los elementos repetidos, mostrando los DIFERENTES oficios de la tabla.

**SELECT DISTINCT OFICIO FROM EMP;**

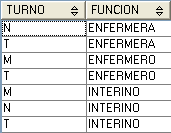


Puedo utilizar Distinct para más de un campo, pero podría perder eficacia dependiendo del caso.

Ejemplo:

* Quiero saber los diferentes turnos que existen para cada función en la plantilla.

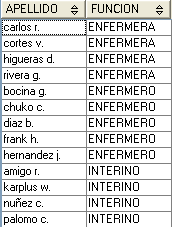
SELECT DISTINCT TURNO, FUNCION FROM PLANTILLA;



**Nota:** Podemos comprobar que las Enfermeras trabajan en el turno de noche y de tarde, que los enfermeros trabajan de tarde y de mañana y que los interinos trabajan en todos los turnos. Ésta consulta puede ser útil.

* Quiero averiguar los diferentes apellidos para cada función de la tabla Plantilla.

SELECT DISTINCT APELLIDO, FUNCION FROM PLANTILLA;



**Nota:** Ésta consulta no tendría sentido, porque no me ofrece ningún dato relevante o de utilidad, ya que quita los apellidos repetidos para cada turno.

**Operador LIKE**

El operador LIKE se utiliza para buscar coincidencias en los campos de texto (VARCHAR) con un valor.

Es un operador que se utiliza sobre todo para los buscadores.

Su coste para el motor de la base de datos es avanzado, ya que no utiliza las hojas de índices (como una guía telefónica) para buscar en los datos, sino, que recorre toda la tabla para recuperar los registros.

Existen dos operadores para realizar las búsquedas con LIKE

* **%** Busca cualquier carácter independientemente de su número de caracteres
* **\_** Busca un único carácter individual.

Ejemplos:

* Buscar todos los empleados de la plantilla cuyo apellido comienza por la letra **a**

SELECT \* FROM PLANTILLA WHERE APELLIDO LIKE 'a%';



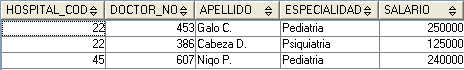
* Buscar todos los empleados de la plantilla cuyo apellido contenga las letras **te**

SELECT \* FROM PLANTILLA WHERE APELLIDO LIKE '%te%';



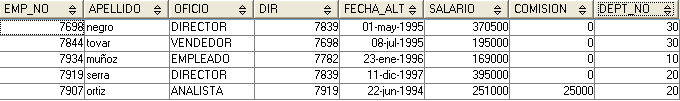
* Buscar todos los doctores cuya especialidad termine con **tria**

SELECT \* FROM DOCTOR WHERE ESPECIALIDAD LIKE '%tria';



* Mostrar los apellidos de los empleados que sean de 5 letras.

SELECT \* FROM EMP WHERE APELLIDO LIKE '\_\_\_\_\_';



**FUNCIONES DE AGRUPACIÓN**

Son funciones que se utilizan para calcular valores en las tablas. Si queremos usarlas combinándolas junto con otros campos debemos utilizar Group by y agrupar los datos que no son funciones.

Con la sentencia group by no se utiliza la cláusula where, se utilizara una clausula propia de la expresión: HAVING. Equivalente a where

* + - **COUNT**: Cuenta los registros que hay en la consulta.

Si pongo un valor dentro de la expresión devolverá la cuenta de todos los registros no nulos.

Si pongo un asterisco contará todos los registros aunque tengan valores nulos.

select count(\*) as “NUMERO” from emp à Valores con Nulos

select count(oficio) as “NUMERO” from emp à Valores sin nulos

* + - **AVG**: Realiza la media sobre la expresión dada, debe ser un tipo de dato Int.

select avg(salario) as “MEDIA” from emp

* + - **MAX**: Saca el valor máximo de una consulta.

select max(fecha\_alt) as “MAXIMO” from emp

* + - **MIN**: Devuelve el valor mínimo de una consulta.

select min(fecha\_alt) as “MINIMO” from emp

* + - **SUM**: Devuelve la suma de los salarios

select sum(salario) as “SUMA” from emp

# Podemos visualizar varias funciones de agrupación dentro de una misma consulta:

select count(\*) as "NUMERO", sum(salario) as "SUMA" from emp;



El problema está cuando queremos mostrar una función de agrupación junto a un campo normal de la tabla.

* + - Pongamos que queremos contar los doctores por especialidad:

SELECT **COUNT(DOCTOR\_NO)** AS "NUMERO", ESPECIALIDAD FROM DOCTOR;

***La sentencia SELECT contiene uno de los errores siguientes:***

*\* En la cláusula SELECT, cláusula HAVING o cláusula ORDER BY están*

*contenidas la expresión identificada y una función de columna, pero*

*no hay ninguna cláusula GROUP BY*

*\* La expresión identificada está contenida en la cláusula SELECT,*

*cláusula HAVING o cláusula ORDER BY, pero no está en la cláusula*

*GROUP BY.*

El error aparece porque las funciones de agrupación, muestran los datos agrupados por el campo o campos que escribamos a partir de la función.

En ésta consulta lo que queremos es mostrar el Número de doctores que existen, pero por **ESPECIALIDAD**.

SELECT **COUNT(DOCTOR\_NO)** AS "NUMERO", ESPECIALIDAD FROM DOCTOR;

Cuando combinamos funciones con campos, debemos utilizar la instrucción

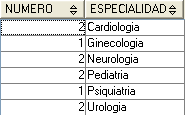
GROUP BY, que agrupa los campos de la consulta.

Ejemplo:

SELECT **COUNT(DOCTOR\_NO)** AS "NUMERO"

, ESPECIALIDAD FROM DOCTOR

**GROUP BY ESPECIALIDAD;**



Lo que nos devuelve es el número de doctores por Especialidad.

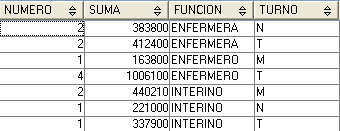
* + - **Debemos incluir dentro del GROUP BY todos los campos que no sean una función de agrupación.**
  + **Queremos mostrar el número de empleados de la plantilla por función y por turno.**

SELECT COUNT(EMPLEADO\_NO) AS "NUMERO"

, SUM(SALARIO) AS "SUMA"

, FUNCION, TURNO FROM PLANTILLA

**GROUP BY FUNCION, TURNO;**



Cuando queremos realizar búsquedas dentro de una consulta con GROUP BY, se puede realizar de dos formas diferentes:

* + - **WHERE**
  + La consulta se realiza sobre **todos** los registros de la tabla, independientemente del GROUP BY.
  + WHERE va después de FROM TABLA.
  + Podemos buscar por cualquier campo de la tabla.
* Vamos a contar el número de doctores por especialidad, pero solamente queremos los doctores de Pediatría.

SELECT COUNT(DOCTOR\_NO) AS "NUMERO"

, ESPECIALIDAD FROM DOCTOR

WHERE ESPECIALIDAD='Pediatria'

GROUP BY ESPECIALIDAD;



* Vamos a contar los doctores por especialidad, pero solo aquellos cuyo salario sea superior a 400.000

SELECT COUNT(DOCTOR\_NO) AS "NUMERO"

, ESPECIALIDAD FROM DOCTOR

WHERE SALARIO > 400000

GROUP BY ESPECIALIDAD;



* Vamos a contar el número de doctores por especialidad, pero sólo mostraremos las especialidades superiores a dos doctores.

SELECT COUNT(DOCTOR\_NO) AS "NUMERO"

, ESPECIALIDAD FROM DOCTOR

**WHERE NUMERO > 2**

GROUP BY ESPECIALIDAD;

**SQL0206N "NUMERO" no es válida en el contexto donde se usa.**

Explicación: No podemos hacer referencia a un “ALIAS” de la columna agrupada.

La sentencia correcta sería:

SELECT COUNT(DOCTOR\_NO) AS "NUMERO"

, ESPECIALIDAD FROM DOCTOR

**WHERE COUNT(DOCTOR\_NO) > 2**

GROUP BY ESPECIALIDAD;

**SQL0120N Uso no válido de una función agregada o de una función OLAP.**

Explicación: WHERE solo puede hacer referencia a columnas de la tabla.

* + - **HAVING**
  + Podemos buscar solamente por los campos de la consulta, agrupados o normales.
  + No podemos buscar por los campos de la tabla.
  + Las consultas son mucho más rápidas por los campos agrupados, ya que primero agrupa y después realiza la búsqueda sobre los campos agrupados. (Muchos menos registros a filtrar).
  + HAVING va después de la sentencia GROUP BY.
* Vamos a contar el número de doctores por especialidad, pero solamente queremos los doctores de Pediatría.

SELECT COUNT(DOCTOR\_NO) AS "NUMERO"

, ESPECIALIDAD FROM DOCTOR

GROUP BY ESPECIALIDAD

HAVING ESPECIALIDAD='Pediatria';



* Vamos a contar los doctores por especialidad, pero solo aquellos cuyo salario sea superior a 400.000

SELECT COUNT(DOCTOR\_NO) AS "NUMERO"

, ESPECIALIDAD FROM DOCTOR

GROUP BY ESPECIALIDAD

HAVING SALARIO > 400000;

**SQL0119N Una expresión que empieza por "SALARIO" especificada en una cláusula**

Explicación: HAVING solamente puede filtrar por los datos de la consulta SELECT.

* Vamos a contar el número de doctores por especialidad, pero sólo mostraremos las especialidades superiores a dos doctores.

SELECT COUNT(DOCTOR\_NO) AS "NUMERO"

, ESPECIALIDAD FROM DOCTOR

GROUP BY ESPECIALIDAD

HAVING NUMERO > 2;

**SQL0206N "NUMERO" no es válida en el contexto donde se usa.**

Explicación: No podemos hacer referencia a un “ALIAS” de la columna agrupada.

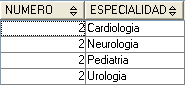
La sentencia correcta sería:

SELECT COUNT(DOCTOR\_NO) AS "NUMERO"

, ESPECIALIDAD FROM DOCTOR

GROUP BY ESPECIALIDAD

HAVING COUNT(DOCTOR\_NO) > 1;



Explicación: con HAVING podemos hacer referencia a las columnas de la consulta y COUNT es una columna de la consulta SELECT.

**Conclusiones**

* + - HAVING: Siempre que necesitemos filtrar por campos contenidos en la clausula SELECT.
    - WHERE: Cuando necesitemos filtrar por columnas de la tabla y que no están contenidas en la clausula SELECT.

**CONSULTAS DE COMBINACIÓN**

**JOIN**

Se usa para combinar resultados entre varias tablas. Es recomendable usar Join ya que consume menos recursos.

Para ver como manejamos este tipo de consultas.

* **Consultas Internas**

Combina las tablas comparando los valores comunes de los campos indicados mediante combinaciones cruzadas.

Sintaxis:

***Select*** *TablaPrincipal.Campo, Tablaconlaquecombinar.Campo*

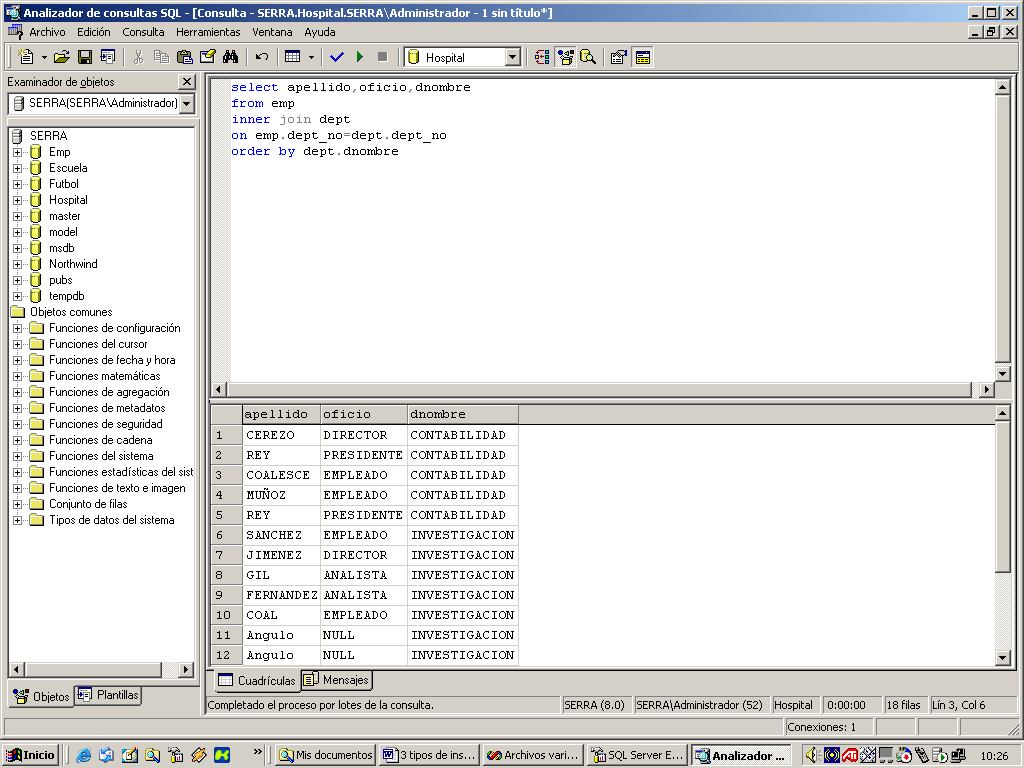
***From*** *TablaPrincipal*

***Inner Join*** *Tablaconlaquecombinar*

## On

*Condición para combinar los campos, es el campo de combinación de las dos tablas.*

* ***Inner Join***: Indica que combine los campos con resultados comunes

****

select apellido,oficio,dnombre

from emp

inner join dept

on emp.dept\_no=dept.dept\_no

order by dept.dnombre

Devuelve todos los Empleados

que tengan asociado un departamento.

* **Otras consultas de combinación:**

Al igual que las consultas de combinación internas, combina los valores comunes de los campos indicados y además de la tabla que queramos, devuelve también el resto de valores aunque no coincidan. Para ello usaremos las siguientes opciones combinadas con join:

Sintaxis:

## Select tablaprincipal.campo, tablaacombinar.campo

*From tablaprincipal*

*left join / right join / cross join tabla*

*on condición*

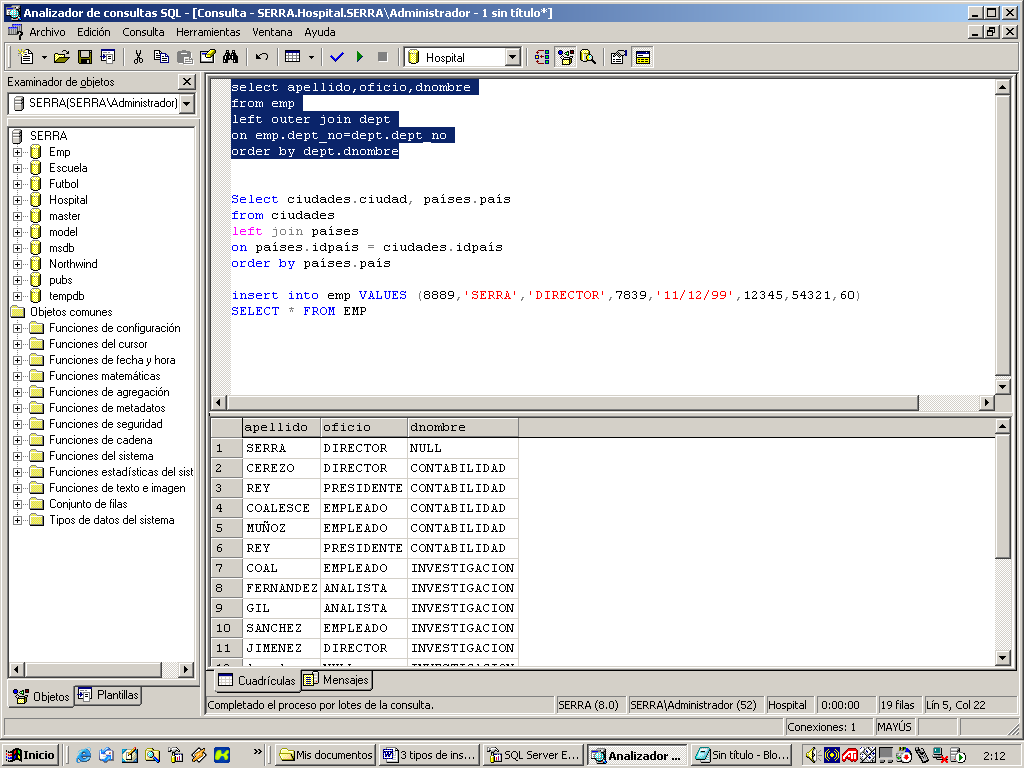
* ***left Join:*** Indica que muestre todos los resultados de la tabla de la izquierda
* ***Right Join:*** Indica que muestre todos los resultados de la tabla de la derecha
* ***Full Join***: Indica que combine todos los campos aunque los resultados sean diferentes.
* ***Cross Join:*** Muestra un producto cartesiano combinando todos los resultados de las dos tablas.

**Nota: La condición para que la tabla sea la izquierda o la derecha es delimitada a partir de la consulta Join, antes del Join (izquierda y después del Join derecha)**

LEFT JOIN

select apellido,oficio,dnombre

from emp



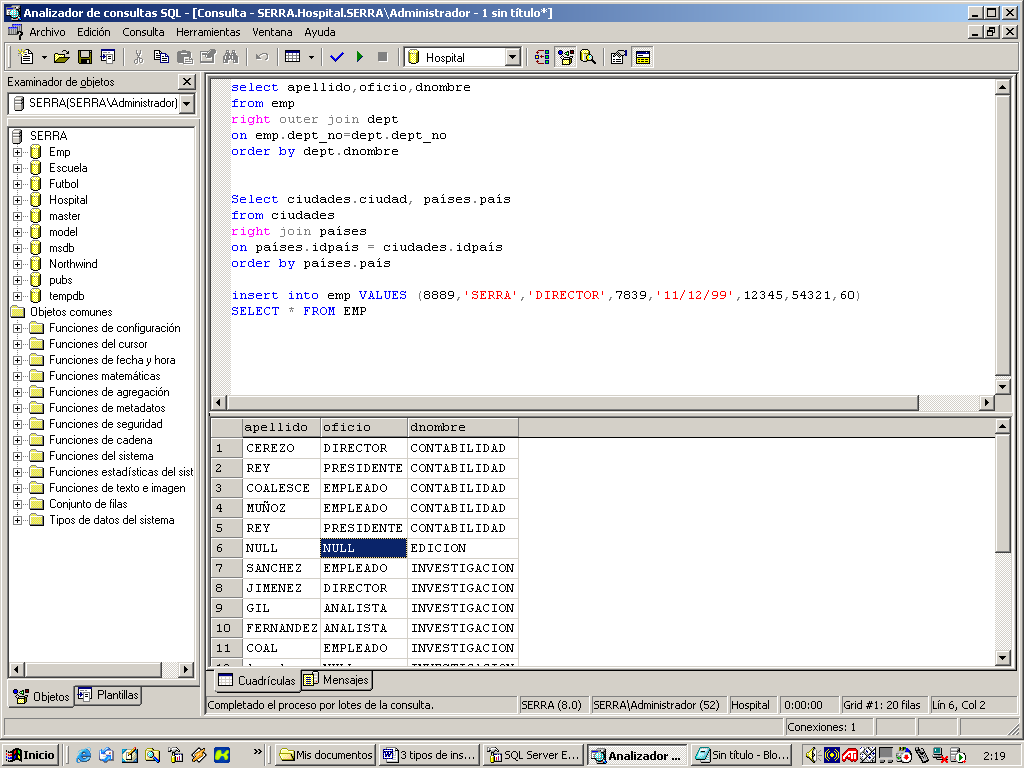
left outer join dept

on emp.dept\_no=dept.dept\_no

order by dept.dnombre

El empleado Serra tiene el nombre del departamento con el valor null porque no tiene ningún departamento asociado y nosotros en la consulta le estamos diciendo que seleccione los empleados aunque no tengan departamento asociado, ponemos como principal la tabla de la izquierda (EMP).

RIGHT JOIN



select apellido,oficio,dnombre

from emp

right outer join dept

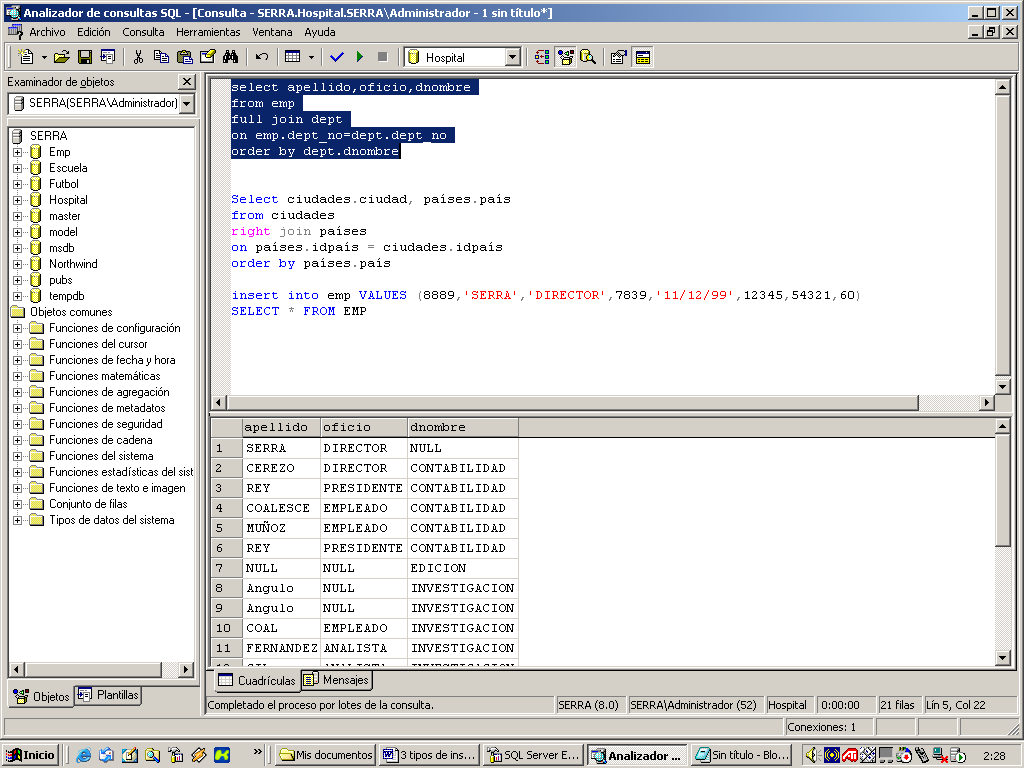
on emp.dept\_no=dept.dept\_no

order by dept.dnombre

En esta consulta el departamento de edición tiene valores null porque le hemos dicho que seleccione la tabla de la derecha como principal (dept), con lo cual selecciona todos los campos de la tabla departamentos con coincidencias con emp o sin ellas.

FULL JOIN

select apellido,oficio,dnombre



from emp

full join dept

on emp.dept\_no=dept.dept\_no

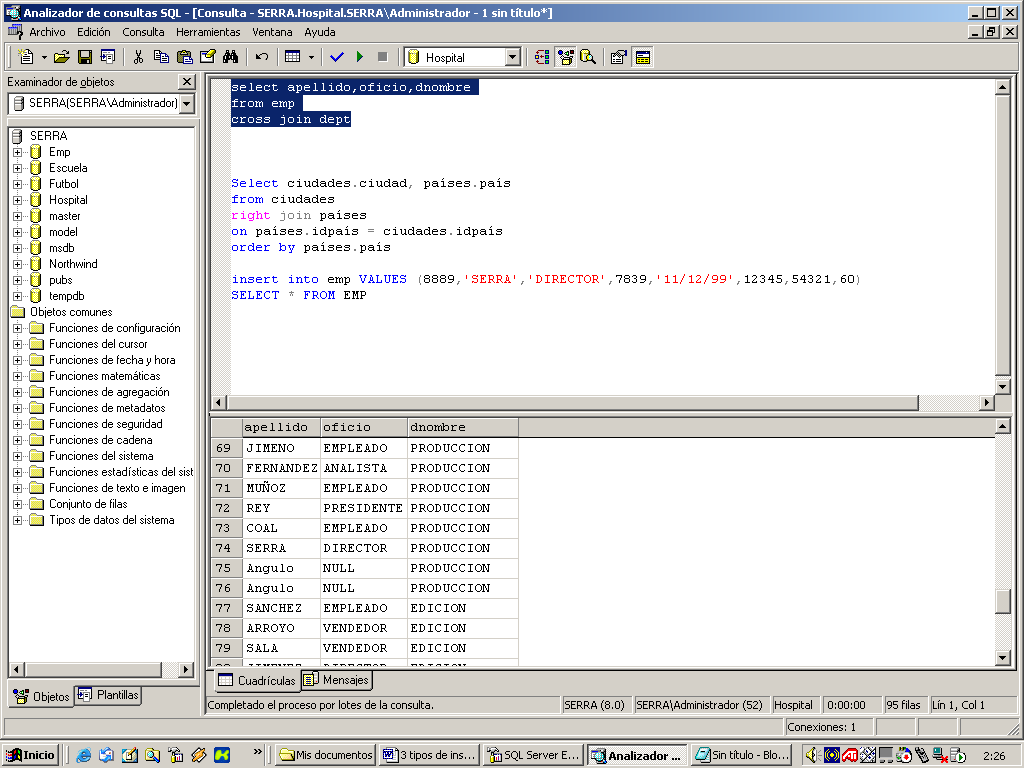
order by dept.dnombre

La combinación **Full Join** muestra las coincidencias de la tabla Dept con Emp, más los valores que no coincidan, como el Empleado SERRA que no tiene departamento y el departamento EDICIÓN, que no tiene empleados.

Se podría decir que es como la suma de utilizar left join y right join.CROSS JOIN

select apellido,oficio,dnombre

from emp



cross join dept

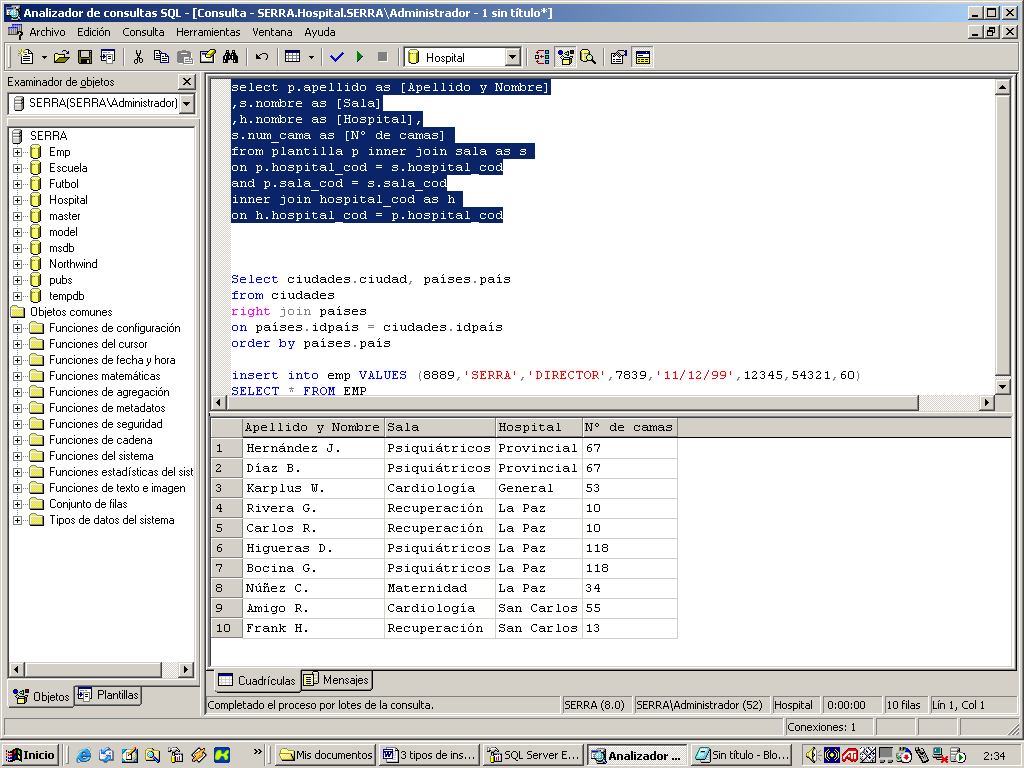
Realiza un producto cartesiano combinando

todos los empleados con todos los

departamentos.

### Combinaciones con mas de dos tablas

Ya hemos visto cómo combinar 2 tablas con inner join, el siguiente ejemplo muestra cómo combinar las 3 tablas que tenemos en la base de datos. Podremos combinar tantas tablas como queramos usando inner join o full join.



select p.apellido as [Apellido]

,s.nombre as [Sala]

,h.nombre as [Hospital],

s.num\_cama as [Nº de camas]

from plantilla p inner join sala as s

on p.hospital\_cod = s.hospital\_cod

and p.sala\_cod = s.sala\_cod

inner join hospital\_cod as h

on h.hospital\_cod = p.hospital\_cod

Podremos usar tantos inner join como queramos en nuestras consultas, pero habrá que tener cuidado a la hora de realizar las combinaciones para que no salgan productos cartesianos en la consulta.

Esta consulta devuelve el nombre del empleado, el nombre de la sala donde trabaja, el nombre del hospital y el número de camas.

OPERADOR UNION

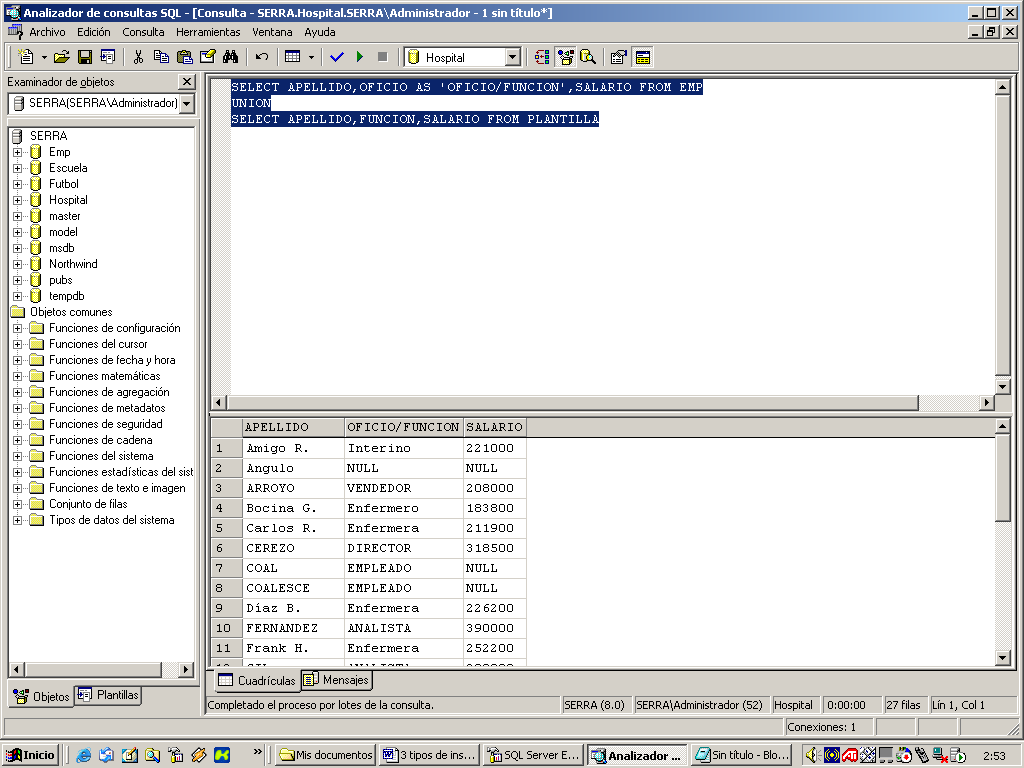
Es un operador que combina un conjunto de resultados, por ejemplo, una sentencia SELECT con OTRA Sentencia SELECT

Sirve para recuperar datos de una o más tablas sin que existe relación entre las tablas.

* No existe límite en el número de tablas a combinar.
* El número de columnas de las consultas debe ser el mismo para todas las consultas
* Los tipo de datos de las columnas deben coincidir en todas las consultas
* Recomendable utilizar alias para las columnas.

Ejemplo:

Recuperar todos los apellidos y oficios de los los empleados y la plantilla.



Select Apellido, Oficio as 'OFICIO/FUNCION'

,salario from emp

UNION

Select Apellido, Funcion, Salario from Plantilla

**SUBCONSULTAS**

* Una subconsulta es una consulta Select que necesita de otras consultas para obtener resultados.
* Las subconsultas deben utilizarse poco, debido a que tiene un gran coste para el motor de la base de datos, ya que bloquea los resultados hasta que no haya devuelto el valor.
* Las subconsultas deben ir anidadas entre paréntesis.
* No hay límite en el momento de utilizar subconsultas, pero su coste para la base de datos aumenta. (Lentitud)

Ejemplo:

Queremos averiguar los empleados que tengan el mismo oficio que SALA

Para realizar ésta consulta, primero debemos saber el oficio de sala, por lo que primero debemos realizar una consulta. Con el resultado de la primera consulta, podemos realizar la segunda.

1. SELECT OFICIO FROM EMP WHERE APELLIDO = ‘SALA’

El resultado de la primera consulta es **VENDEDOR**, ahora podemos utilizarlo.

1. SELECT \* FROM EMP WHERE OFICIO=**’VENDEDOR’**

Una subconsulta realiza ésta acción en una única consulta.

**SELECT \* FROM EMP**

**WHERE OFICIO=**

**(SELECT OFICIO FROM EMP WHERE APELLIDO=’SALA’)**

**Si la consulta devuelve más de un valor, debemos utilizar el operador IN**

Ahora queremos averiguar todos los empleados con el mismo oficio de SALA Y ARROYO.

SELECT \* FROM EMP

WHERE OFICIO=

(SELECT OFICIO FROM EMP WHERE APELLIDO=’SALA’

OR APELLIDO=’ARROYO’)

**Error: La consulta devuelve más de un valor, por lo que debemos utilizar el operador IN para devolver todos los datos.**

SELECT \* FROM EMP

WHERE OFICIO **IN**

(SELECT OFICIO FROM EMP WHERE APELLIDO=’SALA’

OR APELLIDO=’ARROYO’)

### INSERCIÓN, ELIMINACIÓN Y MODIFICACIÓN DE DATOS

**INSERT: Tenemos dos tipos de sintaxis para insertar datos en una tabla.**

* **Inserción de datos parciales**: No introducir todos los datos, solo meter datos en un determinado campo o en varios, pero no en toda la tabla. Los Datos no incluidos, se pondrán a valor **null**.

Sintaxis:

INSERT INTO TABLA (CAMPO1, CAMPO2) VALUES (VALOR1,VALOR2)

Ejemplo:

INSERT INTO PLANTILLA (APELLIDO) VALUES (‘Ramos’)

* **Inserción de todos los datos de una fila:** Se introducen todos los datos de la fila, pero tenemos que corresponder cada posición de la columna con su valor. No es necesario incluir el nombre de las columnas para la consulta.

Sintaxis:

INSERT INTO TABLA VALUES (VALOR1,VALOR2)

Ejemplo:

INSERT INTO PLANTILLA (DEPT)

VALUES (50, ‘MARKETING’, ‘ASTURIAS’)

INSERCION MULTIPLE DE REGISTROS

Sintaxis:

INSERT INTO TABLA (CAMPO1, CAMPO2) VALUES

(DATO1, DATO2)

, (DATO1, DATO2)

, (DATO1, DATO2)

Ejemplo:

INSERT INTO DEPT (DEPT\_NO, DNOMBRE, LOC)

VALUES (50, ‘MARKETING’, ‘SORIA’)

, (60, ‘RRHH’, ‘TERUEL’)

, (70, ‘INFORMATICA’, ‘GRANADA’)

**ELIMINACIÓN DE DATOS**

**DELETE: Elimina una o varias filas de una tabla o vista.**

DELETE FROM {NombreTabla | NombreVista }

WHERE CondicionBusqueda

Delete from emp where apellido = ‘SALA’

# Podemos utilizar subconsultas para la acción, pero solamente un nivel de subconsulta.

# Borrar los de empleados del departamento de informática.

delete from emp

where Dept\_no=

(select Dept\_no from dept dnombre = 'INFORMATICA')

**MODIFICACION DE REGISTROS**

**UPDATE: Modifica una o varias filas de un tabla o vista.**

UPDATE {NombreTabla | NombreVista }

SET CAMPO1 = VALOR1, CAMPO2 = VALOR2

WHERE CondicionBusqueda

Ejemplo: Modificar el departamento de Informática a Ordenadores.

UPDATE DEPT SET DNOMBRE=’ORDENADORES’

WHERE DNOMBRE=’INFORMATICA’

También podemos utilizar subconsultas:

Modificar el salario de los Enfermeros del hospital La paz a 220.000.

update plantilla set salario = 220000

where funcion =’ENFERMERO’

and hospital\_cod=

(select hospital\_cod from hospital

Where nombre=’LA PAZ’)