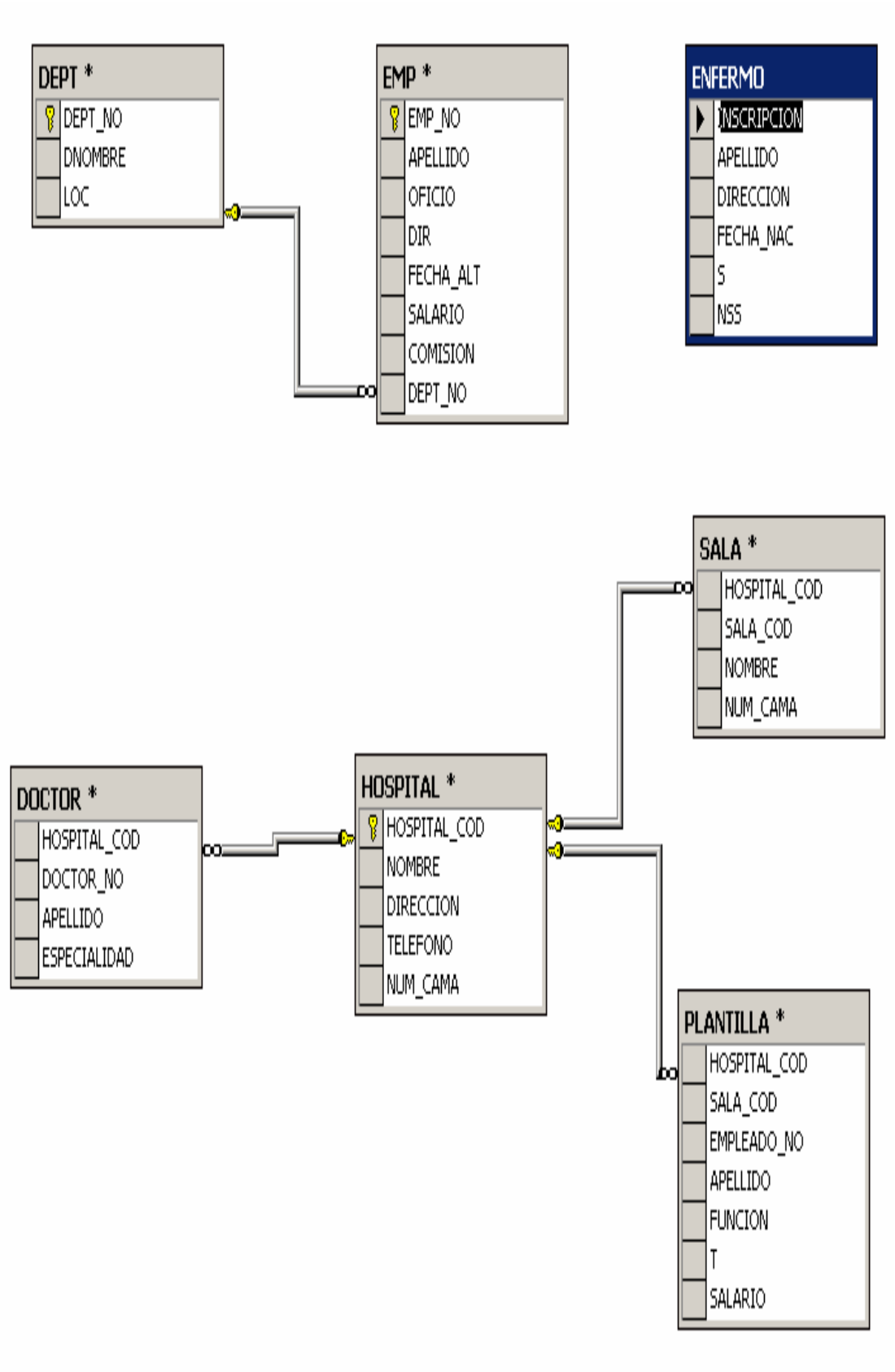
ESTRUCTURA BASE DE DATOS HOSPITAL

RELACIONES ENTRE TABLAS



TIPOS DE DATOS DE LAS COLUMNAS

EMP

	Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
▶	EMP_NO	int	4	✓
	APELLIDO	nvarchar	50	✓
	OFICIO	nvarchar	50	✓
	DIR	int	4	✓
	FECHA_ALT	smalldatetime	4	✓
	SALARIO	int	4	✓
	COMISION	int	4	✓
	DEPT_NO	int	4	✓

DEPT

	Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
	DEPT_NO	int	4	✓
	DNOMBRE	nvarchar	50	✓
	LOC	nvarchar	50	✓

HOSPITAL

	Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
	HOSPITAL_COD	int	4	✓
	NOMBRE	nvarchar	50	✓
	DIRECCION	nvarchar	50	✓
	TELEFONO	nvarchar	50	✓
	NUM_CAMA	int	4	✓

DOCTOR

	Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
	HOSPITAL_COD	int	4	✓
	DOCTOR_NO	int	4	✓
	APELLIDO	nvarchar	50	✓
	ESPECIALIDAD	nvarchar	50	✓

PLANTILLA

	Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
	HOSPITAL_COD	int	4	✓
	SALA_COD	int	4	✓
	EMPLEADO_NO	int	4	✓
	APELLIDO	nvarchar	50	✓
	FUNCION	nvarchar	50	✓
	T	nvarchar	50	✓
	SALARIO	int	4	✓

SALA

	Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
	HOSPITAL_COD	int	4	✓
	SALA_COD	int	4	✓
	NOMBRE	nvarchar	50	✓
	NUM_CAMA	int	4	✓

ENFERMO

	Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
	INSCRIPCION	int	4	✓
	APELLIDO	nvarchar	50	✓
	DIRECCION	nvarchar	50	✓
	FECHA_NAC	smalldatetime	4	✓
	S	nvarchar	50	✓
	NSS	int	4	✓

DATOS DE LA TABLA EMP

	EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
1	7369	SANCHEZ	EMPLEADO	7902	1980-12-17 00:00:00	104000	0	20
2	7499	ARROYO	VENDEDOR	7698	1981-02-22 00:00:00	208000	39000	30
3	7521	SALA	VENDEDOR	689	1981-02-22 00:00:00	162500	65000	30
4	7566	JIMENEZ	DIRECTOR	7839	1981-04-02 00:00:00	386750	0	20
5	7654	MARTIN	VENDEDOR	7698	1981-09-28 00:00:00	182000	182000	30
6	7698	NEGRO	DIRECTOR	7839	1981-05-01 00:00:00	370500	0	30
7	7782	CEREZO	DIRECTOR	7839	1981-06-09 00:00:00	318500	0	10
8	7788	NINO	ANALISTA	7566	1987-03-30 00:00:00	390000	0	20
9	7839	REY	PRESIDENTE	0	1981-11-17 00:00:00	650000	0	10
10	7844	TOVAR	VENDEDOR	7698	1981-09-08 00:00:00	195000	0	30
11	7876	ALONSO	EMPLEADO	7788	1987-05-03 00:00:00	143000	0	20
12	7900	JIMENO	EMPLEADO	7698	1981-12-03 00:00:00	123500	0	30
13	7902	FERNANDEZ	ANALISTA	7566	1981-12-03 00:00:00	390000	0	20
14	7934	MUÑOZ	EMPLEADO	7782	1982-06-23 00:00:00	169000	0	10
15	7119	SERRA	DIRECTOR	7839	1983-11-19 00:00:00	225000	39000	20
16	7322	GARCIA	EMPLEADO	7119	1982-10-12 00:00:00	129000	0	20

DATOS DE LA TABLA DEPT

DEPT_NO	DNOMBRE	LOC
10	CONTABILIDAD	ELCHE
20	INVESTIGACION	MADRID
30	VENTAS	BARCELONA
40	PRODUCCION	SALAMANCA

DATOS DE LA TABLA HOSPITAL

HOSPITAL_COD	NOMBRE	DIRECCION	TELEFONO	NUM_CAMA
19	Provincial	O' Donell 50	964-4256	502
18	General	Atocha s/n	595-3111	987
22	La Paz	Castellana 1000	923-5411	412
45	San Carlos	Ciudad Universitaria	597-1500	845

DATOS DE LA TABLA DOCTOR

HOSPITAL_COD	DOCTOR_NO	APELLIDO	ESPECIALIDAD
22	386	Cabeza D.	Psiquiatría
22	398	Best D.	Urología
19	435	López A.	Cardiología
22	453	Galo D.	Pediatría
45	522	Adams C.	Neurología
18	585	Miller G.	Ginecología
45	607	Chuki P.	Pediatría
18	982	Cajal R.	Cardiología

DATOS DE LA TABLA PLANTILLA

HOSPITAL_COD	SALA_COD	EMPLEADO_NO	APELLIDO	FUNCION	T	SALARIO
22	6	1009	Higueras D.	Enfermera	T	200500
45	4	1280	Amigo R.	Interino	N	221000
19	6	3106	Hernández	Enfermero	T	275000
19	6	3754	Díaz B.	Enfermera	T	226200
22	1	6065	Rivera G.	Enfermera	N	162600
18	4	6357	Karplus W.	Interino	T	337900
22	1	7379	Carlos R.	Enfermera	T	211900
22	6	8422	Bocina G.	Enfermero	M	183800
45	1	8526	Frank H.	Enfermera	T	252200
22	2	9901	Núñez C.	Interino	M	221000

DATOS DE LA TABLA ENFERMO

INSCRIPCION	APELLIDO	DIRECCION	FECHA_NAC	S	NSS
10995	Laguía M.	Goya 20	16-may-56	M	280862422
14024	Fernández M.	Recoletos 50	21-may-60	F	284991452
18004	Serrano V.	Alcalá 12	23-jun-67	F	321790059
36658	Domin S.	Mayor 71	01-ene-42	M	160654471
38702	Neal R.	Orense 11	18-jun-40	F	380010217
39217	Cervantes M.	Perón 38	29-feb-52	M	440294390
59076	Miller B.	López de Hoyos 2	16-sep-45	F	311969044
63827	Ruiz P.	Ezquerdo 103	26-dic-80	M	100973253
64823	Fraiser A.	Soto 3	10-jul-80	F	285201776
74835	Benítez E.	Argentina	05-oct-57	M	154811767

DATOS DE LA TABLA SALA

HOSPITAL_COD	SALA_COD	NOMBRE	NUM_CAMA
22	1	Recuperación	10
45	1	Recuperación	15
22	2	Maternidad	34
45	2	Maternidad	24
19	3	Cuidados Intensivos	21
18	3	Cuidados Intensivos	10
18	4	Cardiología	53
45	4	Cardiología	55
19	6	Psiquiátricos	67
22	6	Psiquiátricos	118

PRÁCTICA Nº__: CONSULTAS DE SELECCIÓN

NOMBRE:

CURSO:

EDICIÓN:

---

1. Mostrar todos los datos de los empleados de nuestra tabla emp.

```
select * from emp
```

2. Mostrar el apellido, oficio, salario anual, con las dos extras para aquellos empleados con comisión mayor de 100000.

```
SELECT APELLIDO, OFICIO,
SALARIO, SALARIO * 14 AS "SALARIO ANUAL" FROM EMP
```

3. Idem del anterior , pero para aquellos empleados que su salario anual con extras supere los 2.200.000 ptas.

```
SELECT APELLIDO, OFICIO,
SALARIO, SALARIO * 14 AS "SALARIO ANUAL" FROM EMP
WHERE SALARIO * 14 > 2200000
```

4. Idem del anterior, pero para aquellos empleados que sumen entre salario anual con extras y comisión los 3.000.000 millones.

```
SELECT APELLIDO, OFICIO,
SALARIO, SALARIO * 14 AS "SALARIO ANUAL" FROM EMP
WHERE SALARIO * 14 + comision > 3000000
```

5. Mostrar todos los datos de empleados ordenados por departamento y dentro de este por oficio para tener una visión jerárquica.

```
select * from emp order by dept_no, oficio
```

6. Mostrar todas las salas para el hospital 45.

```
select * from sala where hospital_cod = 45
```

7. Mostrar todos los enfermos nacidos antes de 1970.

```
select * from enfermo where fecha_nac < '01/01/1970'
```

8. Igual que el anterior, para los nacidos antes de 1970 ordenados por número de inscripción descendente

```
select * from enfermo where fecha_nac < '01/01/1970'
order by inscripcion desc
```

9. Listar todos los datos de la plantilla del hospital del turno de mañana

```
select * from plantilla where T ='M'
```

10. Idem del turno de noche.

```
select * from plantilla where t='N'
```

11. Visualizar los empleados de la plantilla del turno de mañana que tengan un salario entre 200.000 y 225.000 ptas.

```
select * from plantilla where salario between 200000 and 225000
```

12. Visualizar los empleados de la tabla emp que no se dieron de alta entre el 01/01/80 y el 12/12/82.

```
select * from emp where fecha_alt not between '01/01/1980' and
'31/12/1982'
```

13. Mostrar los nombres de los departamentos situados en Madrid o en Barcelona.

```
select DNOMBRE from dept where LOC in ('MADRID','BARCELONA')
```

PRÁCTICA Nº__: CONSULTAS DE SELECCIÓN II

NOMBRE:

CURSO:

EDICIÓN:

---

1. Mostrar aquellos empleados con fecha de alta posterior al 1 de Julio de 1985.

```
select * from emp
where fecha_alt > '01-01-1985'
```

2. Lo mismo que en el ejercicio 1 pero con salario entre 150000 y 400000.

```
select * from emp
where fecha_alt > '01-01-1985'
and salario between 150000 and 4000000
```

3. Igual que en el ejercicio 2, pero también incluimos aquellos que no siendo analista pertenecen al departamento 20.

```
select * from emp
where fecha_alt > '01-01-1985'
and salario between 150000  and 4000000
or (oficio <> 'ANALISTA' and  dept_no = 20)
```

4. Mostrar aquellos empleados cuyo apellido termine en 'Z' ordenados por departamento, y dentro de este por antigüedad.

```
select * from emp
where apellido like '%z'
order by dept_no, fecha_alt asc
```

5. De los empleados del ejercicio 5 quitar aquellos que superen las 200000 ptas mensuales.

```
select * from emp
where apellido like '%z'
and salario > 200000
order by dept_no, fecha_alt asc
```

6. Mostrar todos los empleados cuyo oficio no sea analista.

```
select * from emp
where oficio <> 'ANALISTA'
```

7. Igual que el 6, pero mostrandolos de forma que se aprecien las diferencias de salario dentro de cada oficio.

```
select * from emp
where oficio <> 'ANALISTA'
order by oficio, salario desc
```

8. De los del 7, nos quedamos solo con aquellos cuyo número de empleado no este entre 7600 y 7900.

```
select * from emp
where oficio <> 'ANALISTA'
and emp_no not between 7600 and 7900
order by oficio, salario desc
```

PRÁCTICA Nº__: CONSULTAS DE SELECCIÓN III

NOMBRE:

CURSO:

EDICIÓN:

---

9. Mostrar los distintos oficios de los empleados.

```
select distinct oficio from emp
```

10. Mostrar los distintos nombres de sala.

```
select distinct nombre from sala
```

11. Mostrar que personal "No Interino" existe en cada sala de cada hospital, ordenado por hospital y sala.

```
select hospital_cod, sala_cod, apellido, funcion
from plantilla
where funcion not in ('interino')
order by hospital_cod, sala_cod
```

12. Justificar el resultado de la siguiente consulta SELECT APELLIDO DISTINCT DEPT_NO FROM EMP;. Indicar que ocurre y modificarla para que todo vaya bien.

```
select distinct apellido,dept_no from emp
```

13. Seleccionar los distintos valores del sexo que tienen los enfermos.

```
select distinct s as "SEXO"
from enfermo
```

14. Indicar los distintos turnos de la plantilla del hospital, ordenados por turno y por apellido.

```
select distinct t as "TURNO", apellido
from plantilla
order by turno, apellido
```

15. Seleccionar las distintas especialidades que ejercen los médicos, ordenados por especialidad y apellido.

```
select distinct especialidad, apellido
from doctor
order by especialidad, apellido
```

MANUAL TRANSACT SQL

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 da 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:** Envia lotes de intrucciones 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* ***/**

Tablas en SQL:

- Tabla master: Es la tabla que contiene como generar una base de datos y sobre ella, se crean todas las bases de datos.
- Tabla model: Es la tabla modelo, las bases de datos creadas se basan en esta tabla como modelo.
- Tabla Northwind y Pubs: Son tablas de ejemplos que vienen con SQL y todo usuario puede trabajar con ellas.

➢ **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 #.
Generalmente para las variables locales se usan @ + el nombre.
EJEMPLO: @Contador.
Para las variables totales se usan dos arrobas + el nombre @@Contador
EJEMPLO: @@Error
#Nombre: indica una tabla o procedimiento temporal (Local)
##Nombre: Igual que el anterior pero global.

Tipos de datos:

- **Numéricos:**
  - Enteros → int, tinyint, smallint, bigint
  - Decimales → numeric, decimal, money, smallmoney
  - Coma Flotante → float, real

- **Fechas:**
  - datetime → 0,333 s
  - smalldatetime → 1 minuto

- **Caracteres:**
  - Ancho fijo: char, nchar
  - Ancho Variable: varchar, nvarchar

- **Texto e Imagen:**
  - Text
  - Ntext
  - Rowversion

- **Binario:**
  - Binary, varbinary → Valores tipo byte
  - Bit → 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

OPERADOR LIKE

% Cualquier número de caracteres
_ Para un carácter individual
[ ] Para un conjunto de caracteres que esté dentro del corchete
[^] Que el carácter individual que no esté dentro del corchete

EJEMPLO: **LIKE '%een'** Muestra todos los caracteres que acaben con een
EJEMPLO: **LIKE '%een%'** Muestra todos los caracteres que contengan een en ese orden
EJEMPLO: **LIKE '_en'** Muestra todos los caracteres que contenga tres letras y acaben en en
EJEMPLO: **LIKE '[CK%]'** Muestra todos los caracteres que empiecen por C o K
EJEMPLO: **LIKE '[S-V]ing'** Nombre de 4 letras cuya primera letra estuviera entre S o V y acabe en ing
EJEMPLO: **LIKE 'M[^c]%'** Todos los que empiecen por M y segunda letra no sea una c. No hay limite de caracteres.

CONSULTAS CON LIKE

1. Seleccionar todos los empleados cuyo apellido comience por M

```
select * from emp where apellido like 'M%'
```

	EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
1	7654	MARTIN	VENDEDOR	7698	1981-09-28 00:00:00	182000	182000	30
2	7934	MUÑOZ	EMPLEADO	7782	1982-06-23 00:00:00	169000	0	10

2. Seleccionar todos los empleados cuyo apellido termine con la letra Z

```
select * from emp where apellido like '%z'
```

	EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
1	7369	SANCHEZ	EMPLEADO	7902	1980-12-17 00:00:00	104000	1	20
2	7566	JIMENEZ	DIRECTOR	7839	1981-04-02 00:00:00	386750	0	20
3	7902	FERNANDEZ	ANALISTA	7566	1981-12-03 00:00:00	390000	0	20
4	7934	MUÑOZ	EMPLEADO	7782	1982-06-23 00:00:00	169000	0	10

3. Seleccionar todos los empleados que contengan en su apellido ER.

```
select * from emp where apellido like '%er%'
```

	EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
1	7782	CEREZO	DIRECTOR	7839	1981-06-09 00:00:00	318500	0	10
2	7902	FERNANDEZ	ANALISTA	7566	1981-12-03 00:00:00	390000	0	20

4. Mostrar todos los empleados cuyo nombre sea de 4 letras y su apellido termine con la letra a

```
select * from emp where apellido like '___a'
```

	EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
1	7521	SALA	VENDEDOR	698	1981-02-22 00:00:00	162500	65000	30

5. Mostrar todos los empleados cuyo apellido comience entre las letras E y F.

```
select * from emp where apellido like '[E-F]%'
```

	EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
1	7902	FERNANDEZ	ANALISTA	7566	1981-12-03 00:00:00	390000	0	20

6. Mostrar todos los empleados cuyo apellido comience por la letra A, contenga dentro de su apellido de la letra A a la M y que terminen en O.

```
select * from emp where apellido like 'A%[a-m]%o'
```

	EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
1	8888	Angulo	NULL	NULL	1999-12-12 00:00:00	NULL	NULL	20
2	8888	Angulo	NULL	NULL	1999-12-12 00:00:00	NULL	NULL	20

7. Mostrar todos los empleados cuyo apellido comience por la letra M y la segunda letra no sea una A.

```
select * from emp where apellido like 'M[^A]%'
```

	EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
1	7934	MUÑOZ	EMPLEADO	7782	1982-06-23 00:00:00	169000	0	10

8. Mostrar todos los empleados cuyo apellido sea de 5 letras y su tercera letra sea entra la A y la S terminando en Z.

```
select * from emp where apellido like '__[a-ñ]_z'
```

EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
7934	MUÑOZ	EMPLEADO	7782	1982-06-23 00:00:00	169000	0	10

9. Mostrar todos los empleados cuyo apellido sea de 6 letras y no comience entre la A y la D.

```
select * from emp where apellido like '[^a-d]_____'
```

EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
7654	MARTIN	VENDEDOR	7698	1981-09-28 00:00:00	182000	182000	30
7900	JIMENO	EMPLEADO	7698	1981-12-03 00:00:00	123500	0	30

10. Mostrar todos los que empiecen por la A y cuya cuarta letra no esté comprendida entre A – G

```
select * from emp where apellido like 'A__[^a-g]%'
```

EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
7499	ARROYO	VENDEDOR	7698	1981-02-22 00:00:00	208000	39000	30
8888	Angulo	NULL	NULL	1999-12-12 00:00:00	NULL	NULL	20
8888	Angulo	NULL	NULL	1999-12-12 00:00:00	NULL	NULL	20

Cómo se procesan las consultas no almacenadas en caché (ad hoc)

Analizar = Resolver = Optimizar = Compilar = Ejecutar
→→→→→→→→→
Consultas almacenadas en caché
Primera ejecución

Analizar = Resolver = Optimizar = Compilar = Ejecutar
||
Ejecución Posterior → ||
Procedimiento en Caché = Ejecutar

Cache es una parte de la memoria en la que se almacenan las partes de ejecución de las Consultas.

La mejor forma de almacenar las consultas en caché es con bloques ad hoc:
Lotes ad hoc (Se limitan a coincidencias exactas) :

SELECT * FROM Productos WHERE Precio = 100
SELECT * FROM Productos WHERE Precio = 50
SELECT * FROM Productos WHERE Precio = 100

Otra forma de almacenar consultas en cache Parametrización automatica:
Guarda las constantes que se le pasan con unos parámetros, para la próxima vez que haya una consulta parecida, ya la tiene pero con otros parámetros.

SELECT * FROM Productos WHERE Precio = 100
SELECT * FROM Productos WHERE Precio = 50
SELECT * FROM Productos WHERE Precio = 120

*Esto se hace automáticamente.

Agrupar y resumir datos

Sacar los *n* primeros valores

SELECT TOP n Nombre, Apellido... (Saca los n primeros valores)
SELECT TOP n PERCENT Nombre, Apellido... (Saca el n por ciento)
SELECT TOP n WITH TIES Nombre, Apellido.. ORDER BY (Saca los n primeros ordenados por lo que sea y con coincidencias en el valor n.)

Select top 5 emp_no, apellido from emp → Devuelve los 5 primeros empleados

Select top 50 percent apellido from emp → Devuelve el 50% de los empleados

```
select top 3 with ties oficio, apellido from emp
where oficio = 'EMPLEADO'
order by oficio → Devuelve los 3 primeros registros con oficio
empleado, y si existe algún empleado más, también lo devuelve, porque
coincide con lo que buscamos.
```

Funciones de agregado:

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 clausula 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(*) from emp → Valores con Nulos
select count(oficio) from emp → Valores sin nulos

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

select avg(salario) from emp

- MAX: Saca el valor máximo de una consulta.

select max(fecha_alt) from emp

- MIN: Devuelve el valor mínimo de una consulta.

select min(fecha_alt) from emp

- SUM: Devuelve la suma de los salarios

select sum(salario) from emp

Operadores de SQL:

- Lógicos:

  AND, OR , NOT

- De Comparación:

  = → Igual
  < → Menor
  > → Mayor
  <> → Diferente
  >= → Mayor o igual
  <= → Menor o igual

FUNCIONES DE AGREGADO

1. Encontrar el salario medio de los analistas, mostrando el número de los empleados con oficio analista.

```
select count(*) as [Numero de Empleados], oficio,
avg(salario) as [Salario Medio] from
emp group by oficio having oficio ='ANALISTA'
```

	Numero de Empleados	oficio	Salario Medio
1	3	ANALISTA	335000

2. Encontrar el salario mas alto, mas bajo y la diferencia entre ambos de todos los empleados con oficio EMPLEADO.

```
select  oficio, max(salario) as [Salario mas alto]
, min(salario) as [Salario mas Bajo]
, max(salario) - min(salario) as [Diferencia entre Ambos]
from emp group by oficio having oficio = 'EMPLEADO'
```

	oficio	Salario mas alto	Salario mas Bajo	Diferencia entre Ambos
1	EMPLEADO	169000	100000	69000

3. Visualizar los salarios mayores para cada oficio.

```
select oficio, max(salario) as [Salario Máximo] from emp group by oficio
```

	oficio	Salario Máximo
1	ANALISTA	390000
2	DIRECTOR	390000
3	EMPLEADO	169000
4	PRESIDENTE	650000
5	VENDEDOR	208000

4. Visualizar el número de personas que realizan cada oficio en cada departamento.

```
select dept_no as [Nº de departamento],
count(*) as [Nº de personas], oficio
from emp group by dept_no, oficio
order by 1
```

	N° de departamento	N° de personas	oficio
1	10	1	DIRECTOR
2	10	1	EMPLEADO
3	10	1	PRESIDENTE
4	20	1	VENDEDOR
5	20	2	DIRECTOR
6	20	3	ANALISTA
7	20	3	EMPLEADO
8	30	1	EMPLEADO
9	30	1	DIRECTOR
10	30	4	VENDEDOR
11	50	1	EMPLEADO

5. Buscar aquellos departamentos con cuatro o mas personas trabajando.

```
select dept_no as [Nº de departamento]
, count(*) as [Nº de personas] from emp
group by dept_no having count(*) > 3
```

	N° de departamento	N° de personas
1	20	9
2	30	6

6. Mostrar el número de directores que existen por departamento.

```
select count(*) as [Numero empleados], dept_no from emp
where oficio = 'director'
group by dept_no
```

	Numero empleados	dept_no
1	1	10
2	2	20
3	1	30

7. Visualizar el número de enfermeros, enfermeras e interinos que hay en la plantilla, ordenados por la función.

```
select count(*) as [Nº de personas], FUNCION from plantilla
group by funcion
having funcion in ('ENFERMERO','ENFERMERA','INTERINO')
order by funcion
```

	N° de personas	FUNCION
1	5	Enfermera
2	2	Enfermero
3	3	Interino

8. Visualizar departamentos, oficios y número de personas, para aquellos departamentosque tengan dos o más personas trabajando en el mismo oficio.

```
select dept_no as [Nº de Departamento],count(*) as [Nº de personas],
oficio from emp group by dept_no,oficio  having count(*) > 1
```

	N° de Departamento	N° de personas	oficio
1	20	3	ANALISTA
2	20	2	DIRECTOR
3	20	3	EMPLEADO
4	30	4	VENDEDOR

9. Calcular el salario medio, Diferencia, Máximo y Mínimo de cada oficio. Indicando el oficio y el número de empleados de cada oficio.

```
select oficio, count(*) as [nº de empleados], min(salario) as [Salario mínimo]
, max(salario) as [Salario máximo], max(salario) - min(salario) as [Diferencia]
, avg(salario) as [Media] from emp group by  oficio
```

	oficio	nº de empleados	Salario mínimo	Salario máximo	Diferencia	Media
1	ANALISTA	3	225000	390000	165000	335000
2	DIRECTOR	4	318500	390000	71500	366437
3	EMPLEADO	6	100000	169000	69000	124166
4	PRESIDENTE	1	650000	650000	0	650000
5	VENDEDOR	5	129000	208000	79000	175300

10. Calcular el valor medio de las camas que existen para cada nombre de sala. Indicar el nombre de cada sala y el número de cada una de ellas.

```
select sala_cod as [Sala], Nombre
, avg(num_cama) as [Media de Camas]
from sala group by nombre, sala_cod
```

	Sala	Nombre	Media de Camas
1	1	Recuperación	12
2	2	Maternidad	29
3	3	Cuidados Intensivos	15
4	4	Cardiología	54
5	6	Psiquiátricos	92

11. Calcular el salario medio de la plantilla de la sala 6, según la función que realizan. Indicar la función y el número de empleados.

```
select count(*) as [Nº de empleados], funcion, avg(salario) as [Salario Medio]
from plantilla group by funcion, sala_cod having sala_cod = 6
```

	Nº de empleados	funcion	Salario Medio
1	2	Enfermera	213350
2	2	Enfermero	229400

12. Averiguar los últimos empleados que se dieron de alta en la empresa en cada uno de los oficios, ordenados por la fecha.

```
select max(fecha_alt) as [Fecha], Oficio from emp
group by oficio
order by 1
```

	Fecha	Oficio
1	1981-10-10 00:00:00	DIRECTOR
2	1981-11-17 00:00:00	PRESIDENTE
3	1982-12-11 00:00:00	VENDEDOR
4	1987-03-30 00:00:00	ANALISTA
5	1987-05-03 00:00:00	EMPLEADO

13. Mostrar el número de hombres y el número de mujeres que hay entre los enfermos.

```
select count(*) as [Número],  s as [Sexo] from enfermo group by s
```

	Número	Sexo
1	5	F
2	5	M

14. Mostrar la suma total del salario que cobran los empleados de la plantilla para cada función y turno.

```
select funcion, t as [Turno], sum(salario) as [Suma de Salarios]
from plantilla group by funcion, t
```

15. Calcular el número de salas que existen en cada hospital.

```
select count(*) as [Nº Salas], hospital_cod from sala
group by hospital_cod
```

16. Mostrar el número de enfermeras que existan por cada sala.

```
select count(*) as [Nº Personas], sala_cod, funcion from plantilla
where funcion='enfermera'
group by sala_cod, Funcion
order by 1
```

CONSULTAS DE COMBINACIÓN

JOIN

Se usa para combinar resultados entre varias tablas. Microsoft recomienda 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 / Full Join *Tablaconlaquecombinar*
On
Condición para combinar los campos

- ***Inner Join***: Indica que combine los campos con resultados comunes
- ***Full Join***: Indica que combine todos los campos aunque los resultados sean diferentes.

```
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.

apellido	oficio	dnombre
CEREZO	DIRECTOR	CONTABILIDAD
REY	PRESIDENTE	CONTABILIDAD
COALESCE	EMPLEADO	CONTABILIDAD
MUÑOZ	EMPLEADO	CONTABILIDAD
REY	PRESIDENTE	CONTABILIDAD
SANCHEZ	EMPLEADO	INVESTIGACION
JIMENEZ	DIRECTOR	INVESTIGACION
GIL	ANALISTA	INVESTIGACION
FERNANDEZ	ANALISTA	INVESTIGACION
COAL	EMPLEADO	INVESTIGACION
Angulo	NULL	INVESTIGACION

apellido	oficio	dnombre
SERRA	DIRECTOR	NULL
CEREZO	DIRECTOR	CONTABILIDAD
REY	PRESIDENTE	CONTABILIDAD
COALESCE	EMPLEADO	CONTABILIDAD
MUÑOZ	EMPLEADO	CONTABILIDAD
REY	PRESIDENTE	CONTABILIDAD
NULL	NULL	EDICION
Angulo	NULL	INVESTIGACION
Angulo	NULL	INVESTIGACION
COAL	EMPLEADO	INVESTIGACION
FERNANDEZ	ANALISTA	INVESTIGACION

```
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.

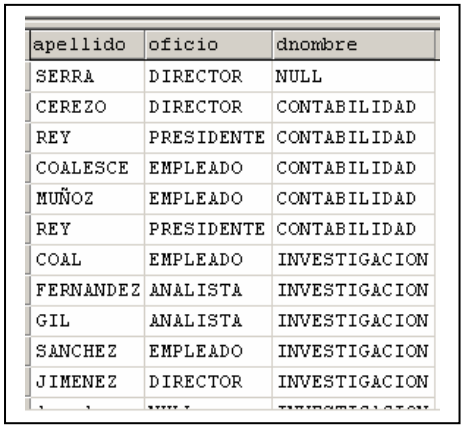
Se podría decir que es como la suma de utilizar left join y right join.

- **Consultas Externas**

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

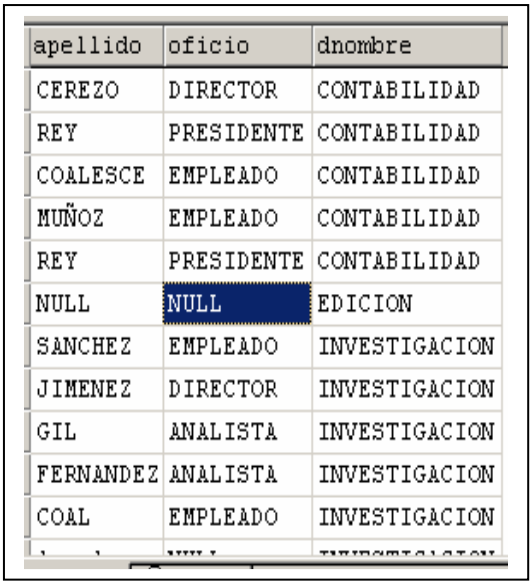
- ***left Join:*** Indica que muestre todos los resultados de la columna de la izquierda
- ***Right Join:*** Indica que muestre todos los resultados de la columna de la derecha
- ***Cross Join:*** Muestra un producto cartesiano combinando todos los resultados de las dos tablas.

```
select apellido,oficio,dnombre
from emp
left outer join dept
on emp.dept_no=dept.dept_no
order by dept.dnombre
```



apellido	oficio	dnombre
SERRA	DIRECTOR	NULL
CEREZO	DIRECTOR	CONTABILIDAD
REY	PRESIDENTE	CONTABILIDAD
COALESCE	EMPLEADO	CONTABILIDAD
MUÑOZ	EMPLEADO	CONTABILIDAD
REY	PRESIDENTE	CONTABILIDAD
COAL	EMPLEADO	INVESTIGACION
FERNANDEZ	ANALISTA	INVESTIGACION
GIL	ANALISTA	INVESTIGACION
SANCHEZ	EMPLEADO	INVESTIGACION
JIMENEZ	DIRECTOR	INVESTIGACION

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).



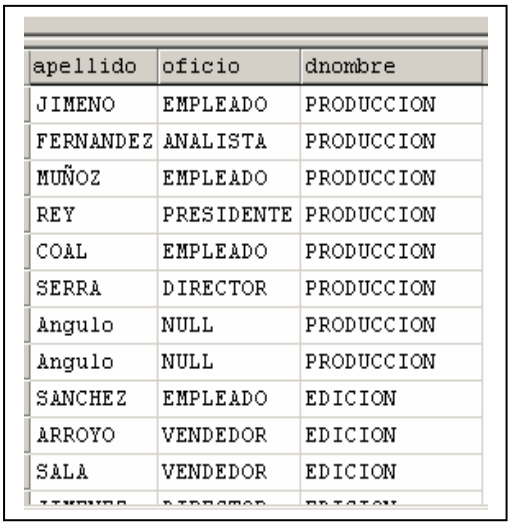
apellido	oficio	dnombre
CEREZO	DIRECTOR	CONTABILIDAD
REY	PRESIDENTE	CONTABILIDAD
COALESCE	EMPLEADO	CONTABILIDAD
MUÑOZ	EMPLEADO	CONTABILIDAD
REY	PRESIDENTE	CONTABILIDAD
NULL	NULL	EDICION
SANCHEZ	EMPLEADO	INVESTIGACION
JIMENEZ	DIRECTOR	INVESTIGACION
GIL	ANALISTA	INVESTIGACION
FERNANDEZ	ANALISTA	INVESTIGACION
COAL	EMPLEADO	INVESTIGACION

```
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.

```
select apellido,oficio,dnombre
from emp
cross join dept
```

Realiza un producto cartesiano combinando todos los empleados con todos los departamentos.



apellido	oficio	dnombre
JIMENO	EMPLEADO	PRODUCCION
FERNANDEZ	ANALISTA	PRODUCCION
MUÑOZ	EMPLEADO	PRODUCCION
REY	PRESIDENTE	PRODUCCION
COAL	EMPLEADO	PRODUCCION
SERRA	DIRECTOR	PRODUCCION
Angulo	NULL	PRODUCCION
Angulo	NULL	PRODUCCION
SANCHEZ	EMPLEADO	EDICION
ARROYO	VENDEDOR	EDICION
SALA	VENDEDOR	EDICION

Combinaciones con mas de dos tablas

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

Apellido y Nombre	Sala	Hospital	Nº de camas
Hernández J.	Psiquiátricos	Provincial	67
Díaz B.	Psiquiátricos	Provincial	67
Karplus W.	Cardiología	General	53
Rivera G.	Recuperación	La Paz	10
Carlos R.	Recuperación	La Paz	10
Higueras D.	Psiquiátricos	La Paz	118
Bocina G.	Psiquiátricos	La Paz	118
Núñez C.	Maternidad	La Paz	34
Amigo R.	Cardiología	San Carlos	55
Frank H.	Recuperación	San Carlos	13

```
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.

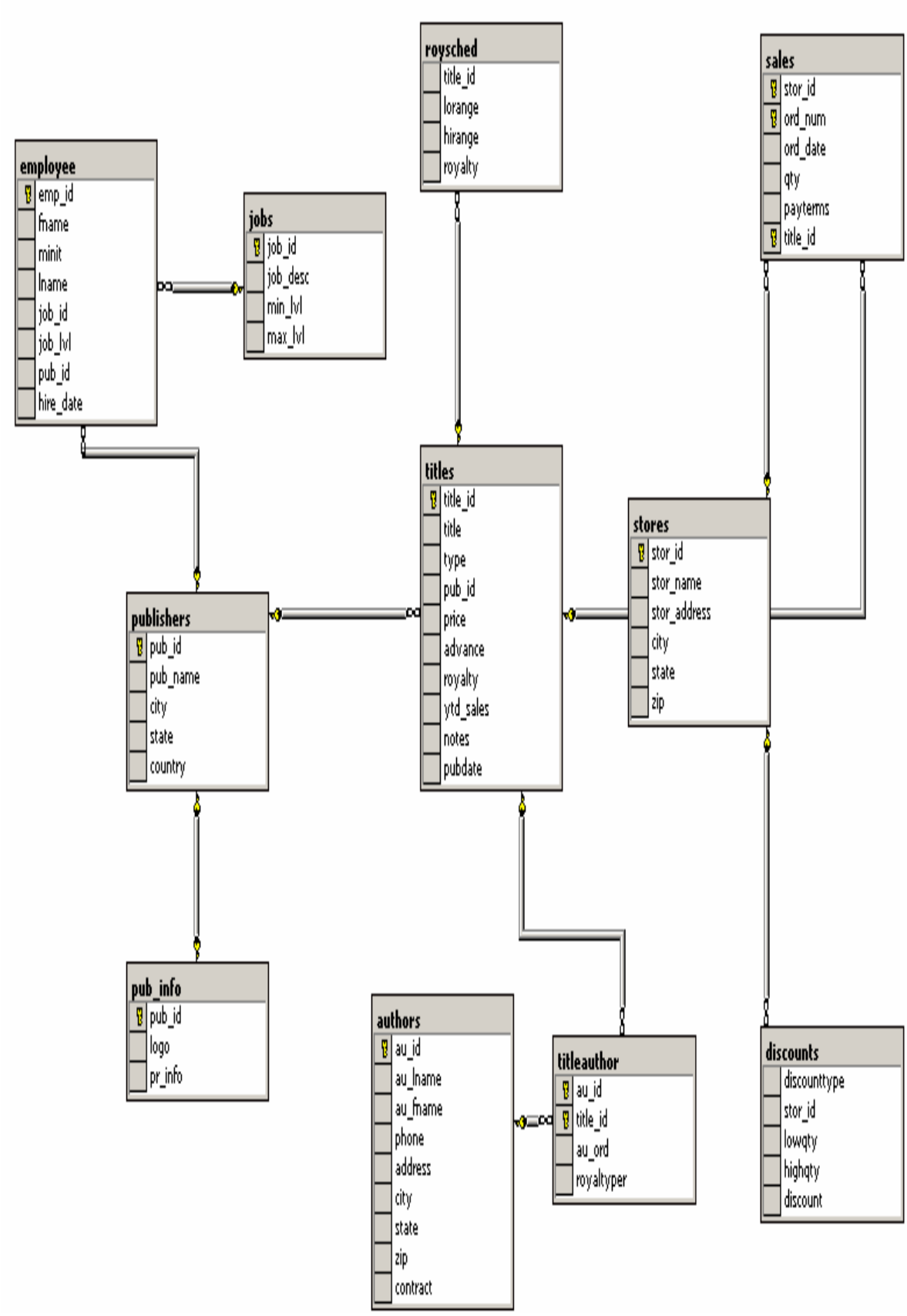
Combinar los valores de una tabla sobre sí misma

Para ello crearemos dos copias de la misma tabla poniéndole un alías, para posteriormente combinar los resultados de ambas copias.

```
select a.emp_no as [Primer Empleado]
,a.apellido,a.dept_no
,b.emp_no as [Segundo Empleado],
b.apellido
from emp as a inner join emp as b
on a.emp_no = b.emp_no
order by b.emp_no
```

Primer Empleado	apellido	dept_no	Segundo Empleado	apellido
7369	SANCHEZ	20	7369	SANCHEZ
7499	ARROYO	30	7499	ARROYO
7521	SALA	30	7521	SALA
7566	JIMENEZ	20	7566	JIMENEZ
7654	MARTIN	30	7654	MARTIN
7698	NEGRO	30	7698	NEGRO
7782	CEREZO	10	7782	CEREZO
7788	GIL	20	7788	GIL
7839	REY	10	7839	REY
7839	REY	10	7839	REY

DIAGRAMA DE LA TABLA PUBS



EJERCICIOS CON COMBINACIONES INTERNAS Y EXTERNAS

1) Realizar una consulta que muestre los nombres de los autores y editores ubicados en la misma ciudad

```
NOMBRE               APELLIDO                                EDITOR
-------------------- --------------------------------------- ----------------------------------------
Cheryl               Carson                                  Algodata Infosystems
Abraham              Bennet                                  Algodata Infosystems

(2 filas afectadas)
```

SOLUCION:

```
use pubs
select a.au_fname as [NOMBRE]
,a.au_lname as [APELLIDO]
,p.pub_name as [EDITOR]
from authors as a
inner join publishers as p
on
a.city = p.city
```

2) Obtener todos los nombres y editores de todos los libros cuyos anticipos pagados son superiores a 7500

```
TITULO                                                                          EDITOR                         ANTICIPO
------------------------------------------------------------------------------- ------------------------------ ---------------------
You Can Combat Computer Stress!                                                 New Moon Books                  10125.0000
The Gourmet Microwave                                                           Binnet & Hardley                15000.0000
Secrets of Silicon Valley                                                       Algodata Infosystems            8000.0000
Sushi, Anyone?                                                                  Binnet & Hardley                8000.0000

(4 filas afectadas)
```

SOLUCION:

```
use pubs
select t.title as [TITULO]
,p.pub_name as [EDITOR]
,t.advance as [ANTICIPO]
from titles as t
inner join publishers as p
on
t.pub_id=p.pub_id
where
t.advance>7500
```

3) Seleccionar todos los titulos, nombre y apellidos del autor de todos los libros de cocina tradicional.

```
NOMBRE                                   APELLIDO             TITULO
---------------------------------------- -------------------- ------------------------------------------------------------
Panteley                                 Sylvia    Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean
Blotchet-Halls                           Reginald             Fifty Years in Buckingham Palace Kitchens
O'Leary                                  Michael              Sushi, Anyone?
Gringlesby                               Burt                 Sushi, Anyone?
Yokomoto                                 Akiko                Sushi, Anyone?

(5 filas afectadas)
```

SOLUCION:

```
select a.au_lname as [NOMBRE]
,a.au_fname as [APELLIDO]
,t.title as [TITULO]
from authors as a
inner join titleauthor as ta
on
a.au_id=ta.au_id
inner join titles as t
on
ta.title_id=t.title_id
where t.type= 'trad_cook'
```

4) Seleccione nombre, apellido de los autores y el nombre de la editorial de todos aquellos escritores cuya ciudad sea la misma que la de la editorial. Pero en la consulta también se incluirán los demás autores de la tabla authors

NOMBRE	APELLIDO	EDITOR
White	Johnson	NULL
Green	Marjorie	NULL
Carson	Cheryl	Algodata Infosystems
O'Leary	Michael	NULL
Straight	Dean	NULL
Smith	Meander	NULL
Bennet	Abraham	Algodata Infosystems
Dull	Ann	NULL
Gringlesby	Burt	NULL
Locksley	Charlene	NULL
Greene	Morningstar	NULL
Blotchet-Halls	Reginald	NULL
Yokomoto	Akiko	NULL
del Castillo	Innes	NULL
DeFrance	Michel	NULL
Stringer	Dirk	NULL
MacFeather	Stearns	NULL
Karsen	Livia	NULL
Panteley	Sylvia	NULL
Hunter	Sheryl	NULL
McBadden	Heather	NULL
Ringer	Anne	NULL
Ringer	Albert	NULL

(23 filas afectadas)

SOLUCION:

```
select a.au_lname as [NOMBRE]
,a.au_fname as [APELLIDO]
,p.pub_name as [EDITOR]
from authors as a
left join publishers as p
on
a.city=p.city
```

5) Recuperar los títulos y el índice del almacén de todos los libros que vendieron más de 25 unidades.

```
ALMACEN ID TITULO
---------- --------------------------------------------------------------------------------
7066       Secrets of Silicon Valley
7066       Is Anger the Enemy?
7067       Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean
7896       You Can Combat Computer Stress!
8042       But Is It User Friendly?

(5 filas afectadas)
```

SOLUCION:

```
select s.stor_id as [ALMACEN ID]
,t.title as [TITULO]
from titles as t
inner join sales as s
on
t.title_id = s.title_id
where
s.qty>25
```

6) Modificación al ejercicio anterior: incluir también los títulos de aquellos libros que no superaron las 25 unidades en sus ventas

```
ALMACEN ID TITULO
---------- --------------------------------------------------------------------------------
8042       But Is It User Friendly?
NULL         Computer Phobic AND Non-Phobic Individuals: Behavior Variations
NULL         Cooking with Computers: Surreptitious Balance Sheets
NULL         Emotional Security: A New Algorithm
NULL         Fifty Years in Buckingham Palace Kitchens
7066       Is Anger the Enemy?
NULL         Life Without Fear
NULL         Net Etiquette
7067       Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean
NULL         Prolonged Data Deprivation: Four Case Studies
7066       Secrets of Silicon Valley
NULL         Silicon Valley Gastronomic Treats
NULL         Straight Talk About Computers
NULL         Sushi, Anyone?
NULL         The Busy Executive's Database Guide
NULL         The Gourmet Microwave
NULL         The Psychology of Computer Cooking
7896       You Can Combat Computer Stress!

(18 filas afectadas)
```

SOLUCION:

```
select s.stor_id as [ALMACEN ID]
,t.title as [TITULO]
from titles as t
left join sales   as s
on
t.title_id = s.title_id and s.qty>25
```

7) Realizar una consulta que devuelva el titulo, editorial y autor de cada libro.

TITULO	AUTOR	EDITOR
The Busy Executive's Database Guide	Green	Algodata Infosystems
The Busy Executive's Database Guide	Bennet	Algodata Infosystems
Cooking with Computers: Surreptitious Balance Sheets	O'Leary	Algodata Infosystems
Cooking with Computers: Surreptitious Balance Sheets	MacFeather	Algodata Infosystems
You Can Combat Computer Stress!	Green	New Moon Books
Straight Talk About Computers	Straight	Algodata Infosystems
Silicon Valley Gastronomic Treats	del Castillo	Binnet & Hardley
The Gourmet Microwave	DeFrance	Binnet & Hardley
The Gourmet Microwave	Ringer	Binnet & Hardley
But Is It User Friendly?	Carson	Algodata Infosystems
Secrets of Silicon Valley	Dull	Algodata Infosystems
Secrets of Silicon Valley	Hunter	Algodata Infosystems
Net Etiquette	Locksley	Algodata Infosystems
Computer Phobic AND Non-Phobic Individuals: Behavior Variations	MacFeather	Binnet & Hardley
Computer Phobic AND Non-Phobic Individuals: Behavior Variations	Karsen	Binnet & Hardley
Is Anger the Enemy?	Ringer	New Moon Books
Is Anger the Enemy?	Ringer	New Moon Books
Life Without Fear	Ringer	New Moon Books
Prolonged Data Deprivation: Four Case Studies	White	New Moon Books
Emotional Security: A New Algorithm	Locksley	New Moon Books
Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean	Panteley	Binnet & Hardley
Fifty Years in Buckingham Palace Kitchens	Blotchet-Halls	Binnet & Hardley
Sushi, Anyone?	O'Leary	Binnet & Hardley
Sushi, Anyone?	Gringlesby	Binnet & Hardley
Sushi, Anyone?	Yokomoto	Binnet & Hardley

(25 filas afectadas)

SOLUCION:

```
select t.title as [TITULO]
,a.au_lname as [AUTOR]
,p.pub_name as [EDITOR]
from publishers as p
inner join titles as t
on
t.pub_id = p.pub_id
inner join titleauthor as ta
on
ta.title_id = t.title_id
inner join authors as a
on
ta.au_id = a.au_id
```

CONSULTAS DE UNION INTERNAS

1. **Seleccionar el apellido, oficio, salario, numero de departamento y su nombre de todos los empleados cuyo salario sea mayor de 300000**

```
select e.apellido
,e.oficio
,e.salario
,d.dept_no as [Nº DEPT]
,dnombre as [DEPARTAMENTO]
from emp as e
inner join dept as d on
e.dept_no = d.dept_no
where e.salario > 300000
```

	apellido	oficio	salario	Nº DEPT	DEPARTAMENTO
1	CEREZO	DIRECTOR	318500	10	CONTABILIDAD
2	NEGRO	DIRECTOR	370500	30	VENTAS
3	JIMENEZ	DIRECTOR	386750	20	INVESTIGACION
4	GIL	ANALISTA	390000	20	INVESTIGACION
5	FERNANDEZ	ANALISTA	390000	20	INVESTIGACION
6	SERRA	DIRECTOR	390000	20	INVESTIGACION
7	REY	PRESIDENTE	650000	10	CONTABILIDAD

2. **Mostrar todos los nombres de Hospital con sus nombres de salas correspondientes.**

```
select s.nombre as [Nombre Sala]
,h.nombre as [Nombre Hospital]
from sala s inner join hospital h on
s.hospital_cod = h.hospital_cod
```

	Nombre Sala	Nombre Hospital
1	Cuidados Intensivos	Provincial
2	Psiquiátricos	Provincial
3	Cuidados Intensivos	General
4	Cardiología	General
5	Recuperación	La Paz
6	Psiquiátricos	La Paz
7	Maternidad	La Paz
8	Cardiología	San Carlos
9	Recuperación	San Carlos
10	Maternidad	San Carlos

3. **Calcular cuantos trabajadores de la empresa hay en cada ciudad.**

```
select count(e.emp_no) as [Nº de trabajadores]
,d.loc as [Ciudad]
from emp as e
right outer join dept as d
on d.dept_no = e.dept_no
group by d.loc
```

	Nº de trabajadores	Ciudad
1	6	BARCELONA
2	3	ELCHE
3	9	MADRID
4	0	SALAMANCA

4. **Visualizar cuantas personas realizan cada oficio en cada departamento mostrando el nombre del departamento.**

```
select d.dnombre as [Nº Departamento]
,count(*) as [Nº de personas]
,e.oficio
from emp e
right outer join dept d
on e.dept_no = d.dept_no
group by e.dept_no, e.oficio,d.dnombre
```

	Nº Departamento	Nº de personas	oficio
1	PRODUCCION	0	NULL
2	CONTABILIDAD	1	DIRECTOR
3	CONTABILIDAD	1	EMPLEADO
4	CONTABILIDAD	1	PRESIDENTE
5	INVESTIGACION	3	ANALISTA
6	INVESTIGACION	2	DIRECTOR
7	INVESTIGACION	3	EMPLEADO
8	INVESTIGACION	1	VENDEDOR
9	VENTAS	1	DIRECTOR
10	VENTAS	1	EMPLEADO
11	VENTAS	4	VENDEDOR

5. **Contar cuantas salas hay en cada hospital, mostrando el nombre de las salas y el nombre del hospital.**

```
select count(s.nombre) as [Numero de salas]
,s.nombre as [Sala]
,h.nombre as [Hospital]
from sala as s
inner join hospital as h
on h.hospital_cod = s.hospital_cod
group by s.nombre,h.nombre
```

	Numero de salas	Sala	Hospital
1	1	Cardiología	General
2	1	Cuidados Intensivos	General
3	1	Maternidad	La Paz
4	1	Psiquiátricos	La Paz
5	1	Recuperación	La Paz
6	1	Cuidados Intensivos	Provincial
7	1	Psiquiátricos	Provincial
8	1	Cardiología	San Carlos
9	1	Maternidad	San Carlos
10	1	Recuperación	San Carlos

6. **Calcular cuantos trabajadores hay en cada departamento(nombre de departamento)**

```
select count(e.emp_no) as [Nº de trabajadores]
,d.dnombre as [Departamento]
from emp e
right outer join dept d
on d.dept_no = e.dept_no
group by d.dnombre
```

	N° de trabajadores	Departamento
1	3	CONTABILIDAD
2	9	INVESTIGACION
3	0	PRODUCCION
4	6	VENTAS

7. **Buscar aquellos departamentos con cuatro o mas personas trabajando.**

```
select d.dnombre as [Departamento]
,count(*) as [Nº de personas]
from emp e
inner join dept d
on e.dept_no = d.dept_no
group by d.dept_no,d.dnombre
having count(*) >= 4
```

	Departamento	N° de personas
1	INVESTIGACION	9
2	VENTAS	6

8. **Calcular el valor medio de las camas que existen para cada nombre de sala. Indicar el nombre de cada sala y el codigo de cada una de ellas.**

```
select avg(num_cama) as [MEDIA CAMAS]
,nombre
,sala_cod
from sala
group by nombre,sala_cod
```

	MEDIA CAMAS	nombre	sala_cod
1	11	Recuperación	1
2	29	Maternidad	2
3	18	Cuidados Intensivos	3
4	54	Cardiología	4
5	62	Psiquiátricos	6

9. Calcular la media salarial por ciudad.

```
select d.loc as [Ciudad]
,avg(salario) as [Media Salarial]
from emp e inner join dept d
on d.dept_no = e.dept_no
group by d.loc
```

	Ciudad	Media Salarial
1	BARCELONA	206916
2	ELCHE	379166
3	MADRID	250861

10. Mostrar los doctores junto con el nombre de hospital en el que ejercen, la dirección y el teléfono del mismo.

```
select d.apellido
,h.nombre
,h.direccion
,h.telefono
from doctor d
inner join hospital_cod h
on h.hospital_cod = d.hospital_cod
```

	apellido	nombre	direccion	telefono
1	Cabeza D.	La Paz	Castellana 1000	923-5411
2	Best D.	La Paz	Castellana 1000	923-5411
3	López A.	Provincial	O' Donell 50	964-4256
4	Galo D.	La Paz	Castellana 1000	923-5411
5	Adams C.	San Carlos	Ciudad Univeritaria	597-1500
6	Miller G.	General	Atocha s/n	595-3111
7	Nino P.	San Carlos	Ciudad Univeritaria	597-1500
8	Cajal R.	General	Atocha s/n	595-3111

11. Mostrar los nombres de los hospitales junto con el mejor salario de los empleados de cada hospital.

```
select h.nombre as [Hospital]
,max(p.salario) as [SALARIO MAXIMO]
from plantilla p
inner join hospital_cod h
on p.hospital_cod = h.hospital_cod
group by h.nombre
```

	Hospital	SALARIO MAXIMO
1	General	337900
2	La Paz	221000
3	Nino	4343
4	Provincial	275000
5	Prueba	32432
6	San Carlos	252200

12. Visualizar el nombre de los empleados de la plantilla junto con el nombre de la sala, el nombre del hospital y el número de camas libres de cada una de ellas.

```
select p.apellido as [Apellido y Nombre]
,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 as h
on h.hospital_cod = p.hospital_cod
```

	Apellido y Nombre	Sala	Hospital	Nº de camas
1	Hernández J.	Psiquiátricos	Provincial	67
2	Díaz B.	Psiquiátricos	Provincial	67
3	Karplus W.	Cardiología	General	53
4	Rivera G.	Recuperación	La Paz	10
5	Carlos R.	Recuperación	La Paz	10
6	Higueras D.	Psiquiátricos	La Paz	118
7	Bocina G.	Psiquiátricos	La Paz	118
8	Núñez C.	Maternidad	La Paz	34
9	Amigo R.	Cardiología	San Carlos	55
10	Frank H.	Recuperación	San Carlos	15

13. Visualizar el máximo salario, mínimo salario de los empleados dependiendo de la ciudad en la que trabajen. Indicando el número total de trabajadores por ciudad.

```
select count(e.emp_no) as [Nº de trabajadores]
,d.loc as [Ciudad]
,max(e.salario) as [Salario Máximo]
,min(e.salario) as [Salario Mínimo]
from emp e
inner join dept d
on e.dept_no = d.dept_no
group by d.loc
```

	Nº de trabajadores	Ciudad	Salario Máximo	Salario Mínimo
1	6	BARCELONA	370500	123500
2	3	ELCHE	650000	169000
3	9	MADRID	390000	100000

14. Averiguar la combinación de que salas podría haber por cada uno de los hospitales.

```
select s.nombre as [Nombre sala]
,h.nombre as [Nombre Hospital]
from sala as s
cross join hospital as h
```

15. Mostrar el Numero de empleado, apellido, oficio y Nombre del departamento de los empleados, junto al Número de empleado, apellido, oficio y Nombre del departamento de sus subordinados respectivamente, para obtener una visión jerarquica de la empresa.

```
select a.emp_no as [Nº de Empleado]
,a.apellido as [Jefe], a.Oficio
,d.dnombre as [Departamento]
,b.emp_no as [Nº Empleado]
,b.apellido as [Subordinado]
,b.Oficio
,d.dnombre as [Departamento]
from emp as a
inner join emp as b
on b.dir = a.emp_no
inner join dept as d
on a.dept_no = d.dept_no
and b.dept_no = d.dept_no
order by b.dir
```

OPERADOR UNION

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

Saca las filas de los empleados y después saca la de Plantilla

APELLIDO	OFICIO/FUNCION	SALARIO
Amigo R.	Interino	221000
Angulo	NULL	NULL
ARROYO	VENDEDOR	208000
Bocina G.	Enfermero	183800
Carlos R.	Enfermera	211900
CEREZO	DIRECTOR	318500
COAL	EMPLEADO	NULL
COALESCE	EMPLEADO	NULL
Díaz B.	Enfermera	226200
FERNANDEZ	ANALISTA	390000
Frank H.	Enfermera	252200

```
Select Apellido,
Oficio as
'OFICIO/FUNCION'
,salario from emp
UNION
Select Apellido,
Funcion, Salario from
Plantilla
```

Recomendaciones a la hora de usar las combinaciones:

- Combinar tablas en funcion de claves principales y externas
- Limite el número de tablas de las combinaciones.

SUBCONSULTAS

Es una SELECT anidada en una instrucción INSERT, DELETE, SELECT o UPDATE.

- **Como una tabla derivada**

```
Select e.emp_no as Numero
,e.Apellido
from (select emp_no, apellido from emp) as e
```

Numero	Apellido
7369	SANCHEZ
7499	ARROYO
7521	SALA
7566	JIMENEZ
7654	MARTIN
7698	NEGRO
7782	CEREZO
7788	GIL
7839	REY
7844	TOVAR
7861	COALESCE

- **Como una expresión**

```
Select emp_no as [Numero]
,Apellido
,Salario
,(select avg(Salario) from emp)
as Diferencia
from emp
where oficio = 'Empleado'
```

Numero	Apellido	Salario	Diferencia
7369	SANCHEZ	104000	287473
7900	JIMENO	123500	287473
7934	MUÑOZ	169000	287473

- **Para correlacionar datos:**

- Expresión dinámica que cambia en cada fila de una consulta externa
- Es una combinación entre la subconsulta y la fila de la consulta externa.
- Dividen consultas complejas con dos o más consultas simples relacionadas

Subconsulta correlacionada

```
Select apellido, oficio,dept_no
from emp as e
where 20 <
(select dept_no from dept as d
where e.dept_no = d.dept_no
and d.dnombre = 'Ventas')
```

apellido	oficio	dept_no
ARROYO	VENDEDOR	30
SALA	VENDEDOR	30
MARTIN	VENDEDOR	30
NEGRO	DIRECTOR	30
TOVAR	VENDEDOR	30
JIMENO	EMPLEADO	30

Simulacion de una clausula JOIN

Vamos a mostrar los oficios que están en más de un departamento.

```
Select distinct e1.oficio from emp as e1
where e1.oficio in(Select e2.oficio from emp as e2
where e1.dept_no <> e2.dept_no)
```

Se debe utilizar antes una combinación que una subconsulta, la combinación sería asi, dando los mismos resultados:

```
Select distinct e1.Oficio from emp as e1
inner join emp as e2
on e1.oficio = e2.oficio
where e1.dept_no <> e2.dept_no
```

Estos son los dos oficios que están en más de un departamento.

oficio
DIRECTOR
EMPLEADO

Subconsulta para simular una clausula HAVING

```
select e1.apellido,e1.oficio, e1.salario
from emp as e1
where e1.salario >
(select avg(e2.salario) from emp as e2
where e1.oficio = e2.oficio)
```

apellido	oficio	salario
CEREZO	DIRECTOR	318500
NEGRO	DIRECTOR	370500
JIMENEZ	DIRECTOR	386750
MUÑOZ	EMPLEADO	169000
TOVAR	VENDEDOR	195000
ARROYO	VENDEDOR	208000

Esta es la consulta utilizando el HAVING, que es la que deberíamos utilizar antes que una subconsulta de simulación HAVING:

```
SELECT e1.apellido,e1.oficio, e1.salario
FROM emp AS e1
INNER JOIN emp AS e2
ON e1.oficio = e2.oficio
GROUP BY e1.oficio, e1.salario,e1.apellido
HAVING e1.salario > AVG (e2.salario)
```

Clausulas EXISTS Y NOT EXISTS

Comprueba si el dato que buscamos existe en la consulta.

Mostrar los empleados que no tienen departamento:

```
select apellido,fecha_alt,salario
from emp as e
where not exists(select * from dept as o
where e.dept_no = o.dept_no)
```

apellido	fecha_alt	salario
SERRA	1999-12-11 00:00:00	12345

Recomendaciones:

- Utilizar subconsultas para dividir una consulta compleja.
- Utilizar Alias para tablas en subconsultas correlacionadas y combinaciones.
- Utilizar EXISTS en vez de IN

SUBCONSULTAS

1. Mostrar el numero de empleado, el apellido y la fecha de alta del empleado mas antiguo de la empresa

   SELECT emp_no,apellido,
   fecha_alt from emp where fecha_alt = (select min(fecha_alt) from emp)

	emp_no	apellido	fecha_alt
1	7369	SANCHEZ	1980-12-17 00:00:00

2. Mostrar el numero de empleado, el apellido y la fecha de alta del empleado mas modernos de la empresa.

   SELECT emp_no,apellido,
   fecha_alt from emp where fecha_alt = (select max(fecha_alt) from emp)

	emp_no	apellido	fecha_alt
1	7876	ALONSO	1987-05-03 00:00:00

3. Visualizar el apellido y el oficio de los empleados con el mismo oficio que Jiménez.

   select apellido, oficio from emp where
   oficio = (select oficio from emp where apellido = 'JIMENEZ')

	apellido	oficio
1	JIMENEZ	DIRECTOR
2	NEGRO	DIRECTOR
3	CEREZO	DIRECTOR
4	SERRA	DIRECTOR

4. Queremos saber el apellido, oficio, salario y número de departamento de los empleados con salario mayor que el mejor salario del departamento 30.

   Select apellido, oficio, salario, dept_no from emp where salario > (select max (salario) from emp where dept_no = 30)

	apellido	oficio	salario	dept_no
1	JIMENEZ	DIRECTOR	386750	20
2	GIL	ANALISTA	390000	20
3	REY	PRESIDENTE	650000	10
4	FERNANDEZ	ANALISTA	390000	20
5	SERRA	DIRECTOR	390000	20

5. Mostrar el apellido, la función, sala o departamento de todos los empleados que trabajen en la empresa.

```
select e.Apellido, e.Oficio as [Cargo en Empresa], d.dnombre as [Oficina]
from emp as e
inner join dept as d
on e.dept_no = d.dept_no
union
select p.Apellido, p.funcion, s.nombre
from plantilla as p
inner join sala as s
on p.hospital_cod = s.hospital_cod
and p.sala_cod = s.sala_cod
union
select d.Apellido, d.especialidad, h.nombre
from doctor as d
inner join hospital as h
on d.hospital_cod = h.hospital_cod
order by 3
```

6. Averiguar el salario de todos los empleados de la empresa, de forma que se aprecien las diferencias entre ellos.

```
select Apellido, Salario from emp
union
select Apellido, Salario from plantilla
order by 2 desc
```

7. Mostrar apellidos y oficio de los empleados del departamento 20 cuyo trabajo sea el mismo que el de cualquier empleado de ventas.

```
Select apellido, oficio from emp where dept_no = 20 and oficio in
 (select oficio from emp where dept_no =
(select dept_no from dept where dnombre = 'ventas'))
order by 2
```

	apellido	oficio
1	JIMENEZ	DIRECTOR
2	SERRA	DIRECTOR
3	ALONSO	EMPLEADO
4	SANCHEZ	EMPLEADO
5	TRICO	EMPLEADO
6	GARCIA	VENDEDOR

8. Mostrar los empleados que tienen mejor salario que la media de los directores, no incluyendo al presidente.

```
Select * from emp where salario >
(select avg (salario) from emp where oficio = 'DIRECTOR')
and oficio <> 'PRESIDENTE'
```

	EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
1	7566	JIMENEZ	DIRECTOR	7839	1981-04-02 00:00:00	386750	0	20
2	7698	NEGRO	DIRECTOR	7839	1981-05-01 00:00:00	370500	0	30
3	7788	GIL	ANALISTA	7566	1987-03-30 00:00:00	390000	0	20
4	7902	FERNANDEZ	ANALISTA	7566	1981-12-03 00:00:00	390000	0	20
5	7919	SERRA	DIRECTOR	7839	1981-10-10 00:00:00	390000	21000	20

9. Mostrar el apellido, función, salario y código de hospital de los empleados de la plantilla que siendo enfermeros o enfermeras pertenecen al hospital SAN CARLOS.

```
Select apellido, funcion, salario, hospital_cod from plantilla where (funcion
= 'ENFERMERO' or funcion = 'ENFERMERA') and hospital_cod = (select
hospital_cod from hospital where nombre = 'SAN CARLOS')
```

```
Select apellido, funcion, salario, hospital_cod from plantilla where
funcion in ('ENFERMERO','ENFERMERA')
and hospital_cod = (select hospital_cod from hospital
where nombre = 'SAN CARLOS')
```

	apellido	funcion	salario	hospital_cod
1	Frank H.	Enfermera	252200	45

	apellido	funcion	salario	hospital_cod
1	Frank H.	Enfermera	252200	45

10. Visualizar los datos de los hospitales que tienen personal (Doctores) de cardiología.

```
Select * from hospital where hospital_cod in (select hospital_cod from
doctor where especialidad = 'Cardiología')
```

	HOSPITAL_COD	NOMBRE	DIRECCION	TELEFONO	NUM_CAMA
1	19	Provincial	O' Donell 50	964-4256	502
2	18	General	Atocha s/n	595-3111	987

11. Visualizar el salario anual de los empleados de la plantilla del Hospital Provincial y General.

```
Select apellido, funcion, salario * 12 as [SALARIO ANUAL] from plantilla
where hospital_cod in (select hospital_cod from hospital where nombre =
'PROVINCIAL' or nombre= 'GENERAL')
```

	apellido	funcion	SALARIO ANUAL
1	Hernández J.	Enfermero	3300000
2	Díaz B.	Enfermera	2714400
3	Karplus W.	Interino	4054800

12. Mostrar el apellido de los enfermos que nacieron antes que el Señor Miller.

```
Select apellido from enfermo where Fecha_nac < (select fecha_nac from
enfermo where apellido = 'MILLER B.')
```

	apellido
1	Domin S.
2	Neal R.

Variables

- Definidas en una instrucción DECLARE (Con una sola @)
- Valores asignados con instrucción SET o SELECT (Con una sola @)
- Ámbito global o local

Sintaxis: DECLARE @VariableLocal tipodatos,...
SET @VariableLocal = expresión

SELECT @VariableLocal = Nombre from Empleados WHERE Id = 5

BUCLES

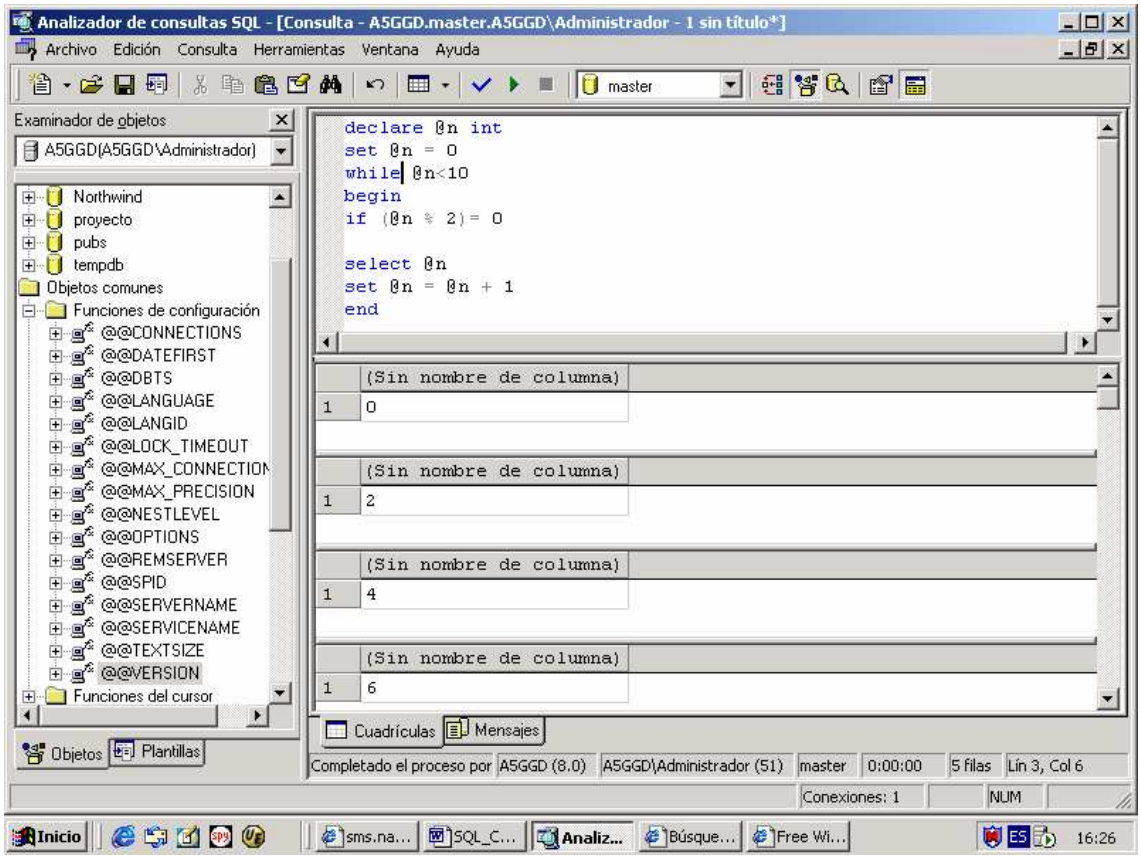
Nivel de instrucción:

- Bloques BEGIN.....END
- Bloques IF.....ELSE
- Construcciones WHILE

Sacar Números pares con IF

```
--IF
declare @n int
set @n = 5
if (@n % 2)= 0
print 'PAR'
else
print 'IMPAR'

--BUCLES
Declare @n int
set @n = 0
while @n < 10
begin
if (@n % 2) = 0
select @n as Numero
set @n = @n + 1
end
```

EJEMPLO: Sacar los numeros pares hasta 10 dentro de un bucle en SQL



Nivel De fila

Las expresiones a nivel de fila evaluan un resultado devuelto por la consulta y dependiendo de los valores que utilicemos lo sustituyen para mejorar la presentación de los datos.

```
CASE expresion
        WHEN valor1 THEN resultado1
        ELSE resultadoN
END

CASE
        WHEN verdadero THEN resultado1
        ELSE resultado2
END
```

```
--CASE
DECLARE @N INT
SET @N = 1
      WHILE (@N<100)
      BEGIN
      SELECT @N AS 'NUMERO', CASE
            WHEN (@N % 2) = 1 THEN
            'IMPAR'
            ELSE
            'PAR'
            END AS 'TIPO'
SET @N = @N + 1
END
```

	NUMERO	TIPO
1	1	IMPAR

	NUMERO	TIPO
1	2	PAR

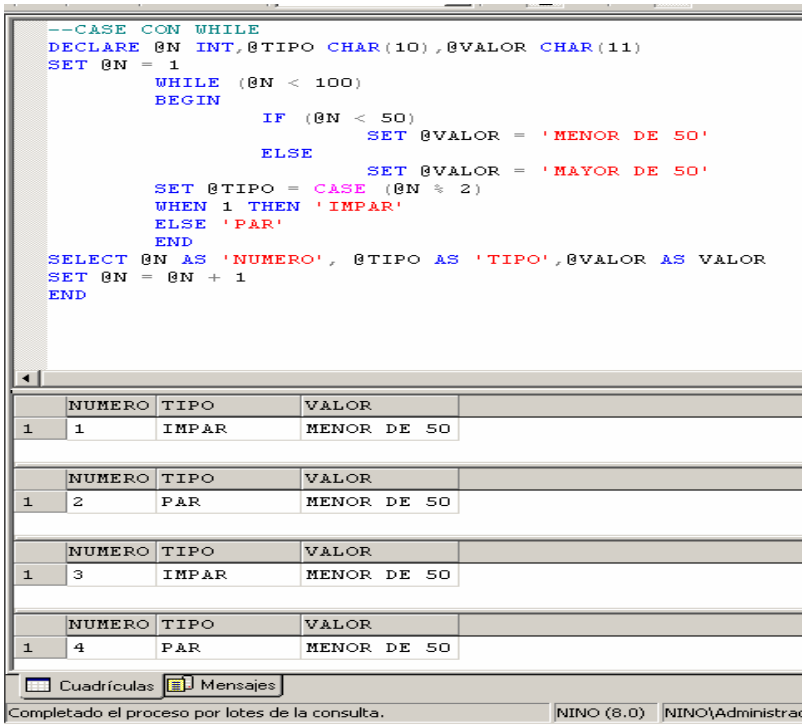
	NUMERO	TIPO
1	3	IMPAR

	NUMERO	TIPO
1	4	PAR

EJEMPLO:

Esto se puede utilizar también para inicializar variables. Es el mismo caso pero utilizando el CASE para inicializar una variable.

```
DECLARE @N INT,@TIPO CHAR(10),@VALOR CHAR(11)
SET @N = 1
      WHILE (@N < 100)
      BEGIN
            IF (@N < 50)
                  SET @VALOR = 'MENOR DE 50'
            ELSE
                  SET @VALOR = 'MAYOR DE 50'
      SET @TIPO = CASE (@N % 2)
      WHEN 1 THEN 'IMPAR'
      ELSE 'PAR'
      END
SELECT @N AS 'NUMERO', @TIPO AS 'TIPO',@VALOR AS VALOR
SET @N = @N + 1
END
```



CONVERT se usa para convertir el numero a una cadena de dos caracteres y lo concatenamos con @tipo.
CAST realiza lo mismo que CONVERT

```
DECLARE @N INT,@TIPO CHAR(10)
SET @N = 5
IF (@N BETWEEN 4 AND 6)
BEGIN
      WHILE (@N > 0)
      BEGIN
      SET @TIPO = CASE (@N % 2)
      WHEN 1 THEN 'IMPAR'
      ELSE 'PAR'
      END
SELECT @N AS 'NUMERO', @TIPO AS 'TIPO'
PRINT CONVERT (CHAR(2),@N) + @TIPO
PRINT CAST (@N AS CHAR(2)) + @TIPO
SET @N = @N - 1
END
END
```

Ponemos Cast y Convert para poder concatenar un valor de tipo Int con un valor de tipo Char, sino intentaría sumarlos y daría error.

EJERCICIOS CON SENTENCIAS

1. Queremos saber a qué empleados eliminariamos si quitasemos los departamentos 10 y 30 y cúales se mantendrían. Mostrar un informe con el apellido, salario, oficio y fechas de alta en la empresa.

```
select apellido,salario,dept_no,oficio,'Accion' = case
when dept_no <> 10 then 'Empleado de baja'
else
'Se Mantiene'
end
from emp where oficio = 'empleado'
```

2. Debemos hacer recortes de salario en la empresa, para ello debemos saber a que personas recortaremos el sueldo, cuales se mantendrán y cuales subiremos el puesto. Utilizar todos los empleados de la empresa(Plantilla y Empleados) Cuando el salario sea menor de 100000, Subiremos sueldo, cuando esté entre 100000 y 250000 lo mantendremos y cuando sea superior, lo bajaremos.

```
select Apellido, Salario, 'Accion' = case
when salario < 100000 then 'Subir sueldo'
when salario between 150000 and 250000 then 'Mantener sueldo'
else
 'Bajar sueldo'
end
from emp
union
select apellido,salario, 'Accion' = case
when salario < 100000 then 'Subir sueldo'
when salario between 150000 and 250000 then 'Mantener sueldo'
else
 'Bajar sueldo'
end
from plantilla
```

3. Queremos saber que empleados de la plantilla trabajan en turno de tarde, noche o en otros, para ello mostraremos ‘Tarde’ o ‘Noche’ dependiendo de sus valores.

```
select 'Número empleado' = empleado_no, 'Apellido' = apellido,
        'Turno' = case t
        when 'T' then 'Tarde'
        when 'm' then 'Mañana'
        else 'Otros'
        end
from plantilla
```

4. Queremos cambiar de localidad en Barcelona, para ello tenemos que saber qué empleados cambiarian de localidad y cúales no. Combinar tablas y mostrar el nombre del departamento junto a los datos del empleado.

```
select d.dnombre as [Departamento]
,e.apellido,'Cambiar de Localidad' = case
when d.loc = 'SEVILLA' THEN 'CAMBIA DE LOCALIDAD'
else
'NO CAMBIA DE LOCALIDAD'
end
from emp as e inner join dept as d
on e.dept_no = d.dept_no
```

5. Queremos saber el número de trabajadores que cambiarían de localidad si cambiasemos a Barcelona y que número de trabajadores no cambiarian de localidad.

```
select count(e.emp_no) as [Nº de trabajadores],
d.loc as [Ciudad],'Cambiar' = case
when d.loc = 'BARCELONA' THEN 'CAMBIA DE LOCALIDAD'
else 'no cambia'
end
from emp as e inner join dept as d
on d.dept_no = e.dept_no
group by d.loc
```

6. Mostrar el apellido, la dirección, la fecha de nacimiento mostrando la década en la que está cada persona y el sexo mostrando si es masculino o femenino de la tabla enfermo.

```
SELECT APELLIDO, DIRECCION,
[FECHA NACIMIENTO] = CASE
WHEN FECHA_NAC BETWEEN '01/01/1940' AND '01/01/1950' THEN
'DECADA DE LOS 40'
WHEN FECHA_NAC BETWEEN '01/01/1950' AND '01/01/1960' THEN
'DECADA DE LOS 50'
WHEN FECHA_NAC BETWEEN '01/01/1960' AND '01/01/1970' THEN
'DECADA DE LOS 60'
WHEN FECHA_NAC BETWEEN '01/01/1970' AND '01/01/1980' THEN
'DECADA DE LOS 70'
WHEN FECHA_NAC BETWEEN '01/01/1980' AND '01/01/1990' THEN
'DECADA DE LOS 80'
ELSE
'DEMASIADO VIEJO O DEMASIADO JOVEN'
END
,SEXO = CASE S
WHEN 'F' THEN 'MUJER'
```

```
ELSE
'HOMBRE'
END
FROM ENFERMO
ORDER BY FECHA_NAC,S
```

7. Mostrar el apellido, el salario, el oficio y el nombre del departamento de todos los empleados aunque no tengan departamento. Si no tienen departamento mostraré que no tienen departamento. Mostraré además si tienen comisión o si no tienen comisión.

```
select e.apellido,e.oficio,e.salario
,DEPARTAMENTO =
ISNULL(d.dnombre,'SIN DEPARTAMENTO')
,Comision = case Comision
when 0 then 'SIN COMISION'
else
'CON COMISION'
end
from emp e
LEFT JOIN DEPT d
ON E.DEPT_NO = D.DEPT_NO
order by d.dnombre
```

8. Mostrar todas las camas que existen para cada hospital y cada sala. Mostraré el nombre del hospital, las salas y su número de camas. Si no hubiese camas para algún hospital las dejaré a 0. También mostraré que son muchas camas cuando sean más de 90, buen número cuando sean mayores de 40 y pocas camas para las demás.

```
select h.nombre as [HOSPITAL]
,isnull(s.num_cama,0) as [Nº DE CAMAS]
,s.nombre as [SALAS]
,CAMAS = CASE
when s.num_cama > 90 then 'DEMASIADAS CAMAS'
when s.num_cama between 40 and 89 then 'BUEN NUMERO'
else
'POCAS CAMAS'
end
from sala as s
full join hospital as h
on h.hospital_cod = s.hospital_cod
group by h.nombre,s.num_cama,s.nombre
order by h.nombre,s.num_cama
```

9. Seleccionar qué empleados están dentro de la media y cuales están por debajo de la media, mostrando el apellido, oficio, salario, comisión y el nombre de los departamentos. No dejar ningún campo a NULL.

```
declare @media int
select @media = avg(salario) from emp
select e.apellido,e.oficio,e.salario,e.comision
,MEDIA = case
when salario > @media then 'DENTRO DE LA MEDIA'
else
'POR DEBAJO DE LA MEDIA'
end
,isnull(d.dnombre,'SIN DEPARTAMENTO')
from emp as e
left join dept d
on e.dept_no = d.dept_no
group by e.apellido,e.oficio,e.salario,e.comision,d.dnombre
order by Media
```

INSERCIÓN, ELIMINACIÓN Y MODIFICACIÓN DE DATOS

- **Inserción de una fila mediante valores:**

  INSERT INTO {NombreTabla | NombreVista} [Valor de la Columna]
  VALUES Valores

* Cuando hay llaves es porque se debe elegir entre uno de los dos, esta barra | indica que se debe poner uno de los dos valores.

- **Uso INSERT...SELECT:**

  INSERT NombreTabla SELECT ListaColumnas FROM ListaTablas
  WHERE CondicionBusqueda

  Se introducen en la tabla las columnas y filas que devuelva con sus respectivos datos. La consulta SELECT debe devolver los datos adecuados para la tabla donde vamos a introducir los valores.

- **Creación de una tabla mediante SELECT INTO:** Creación de una tabla que a la vez se le introducen valores.

  SELECT ListaColumnas INTO NuevaTabla FROM TablaOrigen
  WHERE CondicionBusqueda

  ```
  select apellido,salario,dept_no into #Temporal
  from emp
  where dept_no = 60
  ```

  Se utiliza mucho para crear tablas temporales

- **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.

- **Inserción de datos mediante valores de columna predeterminados: Se usa para no dejar a las tablas con el valor null y así no da error.**

  Se utilizan dos clausulas:

  - DEFAULT: Especificar que cogiera en la lista de valores el valor por defecto de esa columna
  - DEFAULT VALUES: Crea una nueva fila con los valores por defecto de todas las columnas

  ```
  USE Hospital
  INSERT INTO emp  (Apellido,Salario)
  VALUES ('SERRA', DEFAULT)
  ```

Con esta sentencia se pone el valor predeterminado que tenga la tabla, si no tiene valor por defecto, pondrá null, lo que equivale a no poner el dato. Los valores por defecto se verán más adelante.

ELIMINACIÓN DE DATOS

- DELETE: Elimina una o varias filas. Hay un control de las modificaciones (Borrado) que se estan haciendo.

```
DELETE [FROM (Opcional) ] {NombreTabla | NombreVista }
WHERE CondicionBusqueda
Delete from emp where apellido = 'SERRA'
```

- TRUNCATE TABLE: Elimina todas las filas de la tabla (La tabla con su estructura no se elimina, sólo los datos de la tabla). No crea filas en el registro de transacciones, con lo cual es el método más rápido de borrar.

```
TRUNCATE TABLE NombreTabla
Truncate Table emp
```

- Eliminación de filas basada en otras tablas

```
DELETE [ FROM ] {NombreTabla | NombreVista}
[ FROM, OrigenTabla,... ]  [ WHERE CondicionBusqueda ]
```

Borra los campos de emp donde tienen relacion con informática

```
delete from emp
from emp as e
inner join departamento as d
on e.dept_no = d.dept_no
where d.dnombre = 'INFORMATICA'
```

ACTUALIZACIONES

- Actualización de filas basadas en datos de la propia tabla

```
UPDATE {NombreTabla | NombreVista }
SET NombreColumna = expresión { DEFAULT | NULL, ... }
```

```
USE Northwind
UPDATE products
SET unitprice = (unitprice * 1.1 )
```

- Actualización de filas basadas en otras tablas

```
UPDATE {NombreTabla | NombreVista }
SET NombreColumna = expresión { DEFAULT | NULL, ... }
FROM OrigenTabla
WHERE CondicionBusqueda
```

Cambiar el salario de los empleados del dept 30 donde el departamento sea 60.

```
update emp set salario = 130000 from emp
inner join dept
on emp.dept_no = dept.dept_no
where dept.dept_no = 60
```

CONSULTAS DE ACCION

1. Dar de alta con fecha actual al empleado Jose Escriche Barrera como programador perteneciente al departamento de informatica.Tendra un salario base de 70000 pts/mes y no cobrara comision, ¿qué dificultad plantea el alta de este empleado? ¿Cómo podria solucionarse ?

   ```
   Insert into dept(dept_no,dnombre,loc)
   values(60,'INFORMATICA','MADRID')

   Insert into emp(apellido,oficio,fecha_alt,salario,comision,dept_no)
   values('Escriche','PROGRAMADOR','07/02/02',70000,0,60)
   ```

2. Se quiere dar de alta un departamento de informática situado en Fuenlabrada (Madrid).

   ```
   insert into dept(dept_no,dnombre,loc)
   values(70,'INFORMATICA','FUENLABRADA')
   ```

3. El departamento de ventas por motivos de peseteros se traslada a Lerida , realizar dicha modificación.

   ```
   update dept set loc='LERIDA' where DNOMBRE='VENTAS'
   ```

4. En el departamento anterior se dan de alta dos empleados: Julián Romeral y Luis Alonso. Su salario base es de 80000 pts y cobrarán una comisión del 15% de su salario.

   ```
   insert into emp(dept_no,apellido,salario,comision,emp_no)
   values(30, 'Romeral',80000,80000*0.15,7500)

   insert into emp(dept_no,apellido,salario,comision,emp_no)
   values(30, 'Alonso',80000,80000*0.15,7600)
   ```

5. Modificar la comisión de los empleados de la empresa, de forma que todos tengan un incremento del 10% del salario.

   ```
   update emp set salario=salario*1.1
   ```

6. Incrementar un 5% el salario de los interinos de la plantilla que trabajen en el turno de noche.

   ```
   update plantilla set salario = salario*1.05 where funcion='Interino'
   and  T='N'
   ```

7. Incrementar en 5000 pts el salario de los empleados del departamento de ventas y del presidente, tomando en cuenta los que se dieron de alta antes que el presidente de la empresa.

```
update emp set salario = salario + 5000
from emp as e
inner join dept as d
on e.dept_no = d.dept_no
WHERE FECHA_ALT <  (select fecha_alt from emp where oficio =
'PRESIDENTE')
AND e.OFICIO = 'PRESIDENTE'
OR d.dnombre = 'VENTAS'
```

8. Se tienen que desplazar cien camas del Hospital SAN CARLOS para un Hospital de Venezuela. Actualizar el número de camas del Hospital SAN CARLOS.

```
update Hospital  set num_cama = num_cama-100 where nombre='San
Carlos'
```

9. Crear una tabla llamada Mujeres e insertar los enfermos con este sexo.

```
insert into Mujeres select * from enfermo where S = 'F'
```

10. Crear una tabla llamada Empleados e introducir todos los datos de la tabla EMP en ella.

```
INSERT INTO empleados select * from emp
```

11. Utilizar la tabla anterior. Subir el salario y la comisión en un millón de pesetas y doscientas veinticinco mil pesetas respectivamente a los empleados que se dieron de alta en este año.

```
update empleados set salario = salario + 1000000/12,
comision = comision + 225000/12 where fecha_alt > '01/01/02'
```

12. Borrar de la tabla mujer al enfermo con número de inscripción igual a 64823.

```
delete from Mujeres where inscripcion= '64823'
```

13. Borrar todos los registros de la tabla Mujeres de la forma más rápida.

```
truncate table mujeres
```

14. Utilizar la tabla Empleados. Borrar todos los empleados dados de alta entre las fechas 01/01/80 y 31/12/82.

```
delete from empleados where fecha_alt between '01/01/80' and '31/12/82'
```

15. Modificar el salario de los empleados trabajen en la paz y esten destinados a Psiquiatría. Subirles el sueldo 20000 ptas más que al señor Amigo R.

```
update plantilla set salario =
(select salario + 20000 from plantilla
where apellido = 'Amigo R.')
from plantilla as p
inner join hospital as h
on h.hospital_cod = p.hospital_cod
inner join sala as s
on s.sala_cod = p.sala_cod
where h.nombre = 'La Paz'
and s.nombre = 'Psiquiátricos'
```

16. Borrar los empleados cuyo nombre de departamento sea producción.

```
delete from emp
from emp as e
inner join departamento as d
on e.dept_no = d.dept_no
where d.dnombre = 'PRODUCCION'
```

MÓDULO 4: PROCEDIMIENTOS PARA AGRUPAR Y RESUMIR DATOS

En el examen de certificación las bases de datos que se suelen usar son las que vienen de ejemplo en SQL, es decir Northwind y Pubs
Use Base de datos Indica que la siguiente sentencia usará la base de datos indicada.
Ejemplo: Ponemos en el analizador de consultas lo siguiente:

```
Use Hospital
Select * from Hospital
```

[Order Details]

Esta tabla perteneciente a la Northwind, se encarga de manejar los pedidos

Order Id : Nº de Pedido.
Producto Id : Nº de Producto.

Ambos campos son Primary Key, con lo que no puede haber una combinación de ambos campos que sea igual.

OrderId	ProductId
1	A
1	B
1	C
2	A
3	C

Es decir en este caso no podría existir una nueva combinación “1 A” o “2 A”.
Quantity: Es la cantidad del producto del pedido.

ROLLUP

Se usa para presentar resúmenes de datos. A de usarse junto con la clausula group by, lo que hace es realizar un resumen de los campos incluidos en el rollup.

```
Select ProductID, OrderId,
Sum(Quantity) As Cantidad_Total
From [Order Details]
Group by ProductID, OrderID
with Rollup
Order by ProductID, OrderID
```

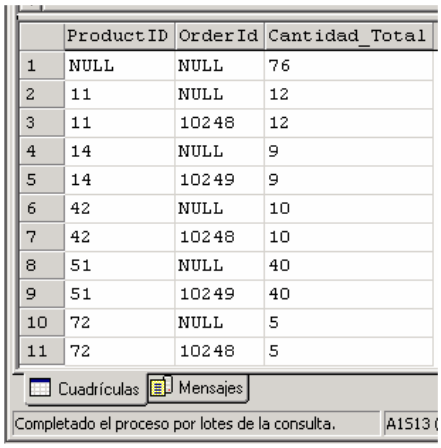
Este ejemplo suma todas las cantidades, y mediante rollup, muestra una fila con la suma de todas las cantidades de cada producto, y además, otra fila con la suma de todas las cantidades de todos los productos. El resultado de este ejemplo, sería el que muestra la imagen:

	ProductID	OrderId	Cantidad_Total
1	NULL	NULL	51317
2	1	NULL	828
3	1	10285	45
4	1	10294	18
5	1	10317	20
6	1	10348	15
7	1	10354	12
8	1	10370	15
9	1	10406	10
10	1	10413	24
11	1	10477	15
12	1	10522	40

Cuadrículas | Mensajes

Completado el proceso por lotes de la consulta. A1513 (8.0

```
Select ProductID, OrderId,
Sum(Quantity) As Cantidad_Total
From [Order Details]
where orderid < 10250
Group by ProductID, OrderID
with Rollup
Order by ProductID, OrderID
```



	ProductID	OrderId	Cantidad_Total
1	NULL	NULL	76
2	11	NULL	12
3	11	10248	12
4	14	NULL	9
5	14	10249	9
6	42	NULL	10
7	42	10248	10
8	51	NULL	40
9	51	10249	40
10	72	NULL	5
11	72	10248	5

Cuadrículas Mensajes

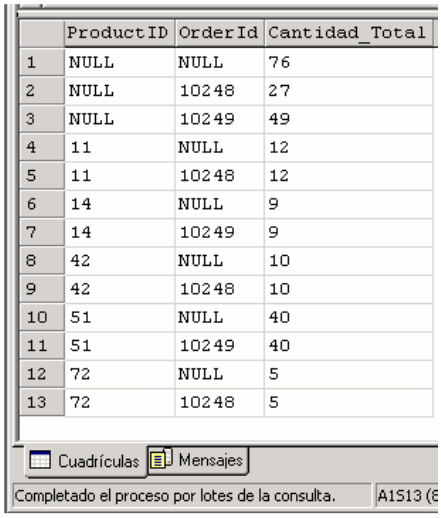
Completado el proceso por lotes de la consulta. A1513 (

CUBE

Al igual que Rollup realiza resúmenes de campos agrupados. Pero en este caso muestra un resumen con cada combinación posible de los campos agrupados.

```
Select ProductID, OrderId,
Sum(Quantity) As Cantidad_Total
From [Order Details]
where orderid < 10250
Group by ProductID, OrderID
with Cube
Order by ProductID, OrderID
```

En este caso como vemos en la imagen, hace un resumen con la suma de la cantidad de cada combinación posible entre el productid y el orderid



	ProductID	OrderId	Cantidad_Total
1	NULL	NULL	76
2	NULL	10248	27
3	NULL	10249	49
4	11	NULL	12
5	11	10248	12
6	14	NULL	9
7	14	10249	9
8	42	NULL	10
9	42	10248	10
10	51	NULL	40
11	51	10249	40
12	72	NULL	5
13	72	10248	5

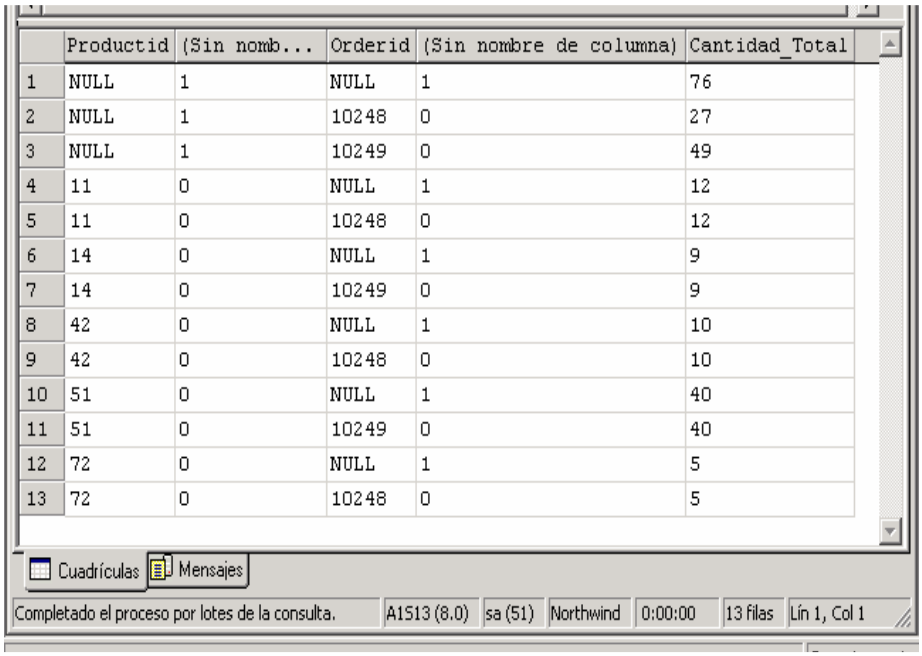
Cuadrículas Mensajes

Completado el proceso por lotes de la consulta. A1513 (

GROUPING

Indica si el resultado de un campo es el que hay en la propia tabla o se ha introducido mediante una cláusula de resumen, es decir, para saber por ejemplo si un “Null” de una celda es de la propia tabla o ha es debido a una clausula Cube o Rollup.

```
Select Productid, grouping(ProductID), Orderid, grouping(OrderId),
Sum(Quantity) As Cantidad_Total
From [Order Details]
where orderid < 10250
Group by ProductID, OrderID
with Cube
Order by ProductID, OrderID
```



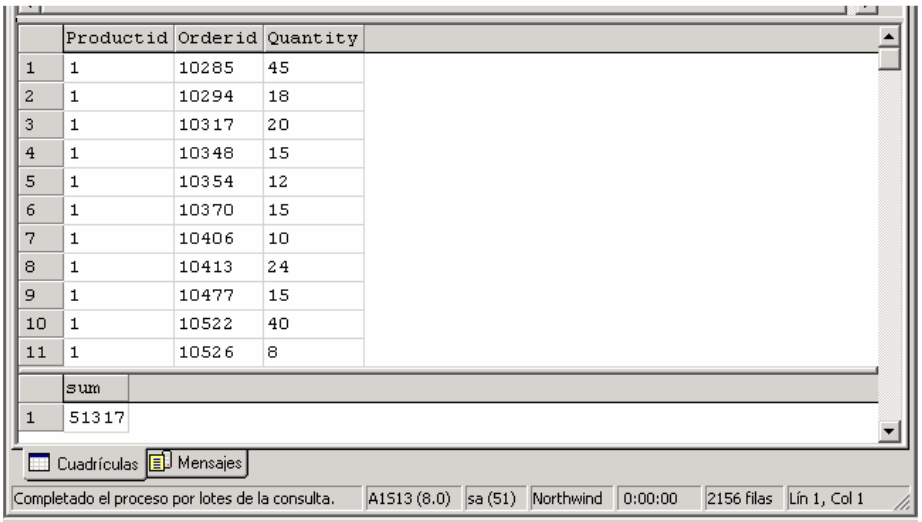
	Productid	(Sin nomb...	Orderid	(Sin nombre de columna)	Cantidad_Total
1	NULL	1	NULL	1	76
2	NULL	1	10248	0	27
3	NULL	1	10249	0	49
4	11	0	NULL	1	12
5	11	0	10248	0	12
6	14	0	NULL	1	9
7	14	0	10249	0	9
8	42	0	NULL	1	10
9	42	0	10248	0	10
10	51	0	NULL	1	40
11	51	0	10249	0	40
12	72	0	NULL	1	5
13	72	0	10248	0	5

Vemos que por cada grouping que hemos puesto, sale una columna, 1 indica que el Null es creado por la clausula Cube y 0 indica que es de la tabla. Poniéndole un alias al grouping saldría el nombre de columna que le indiquemos.

COMPUTE

Realiza un resumen en una columna aparte con el resultado de la función de agregado indicada. Su formato sería Compute función(campo). No se puede utilizar en aplicaciones cliente / servidor, es una clausula meramente informativa. Tampoco se le pueden poner alias a los resúmenes.

```
Select Productid, Orderid, Quantity
From [Order Details]
order by ProductID, Orderid
compute Sum(quantity)
```



	Productid	Orderid	Quantity
1	1	10285	45
2	1	10294	18
3	1	10317	20
4	1	10348	15
5	1	10354	12
6	1	10370	15
7	1	10406	10
8	1	10413	24
9	1	10477	15
10	1	10522	40
11	1	10526	8

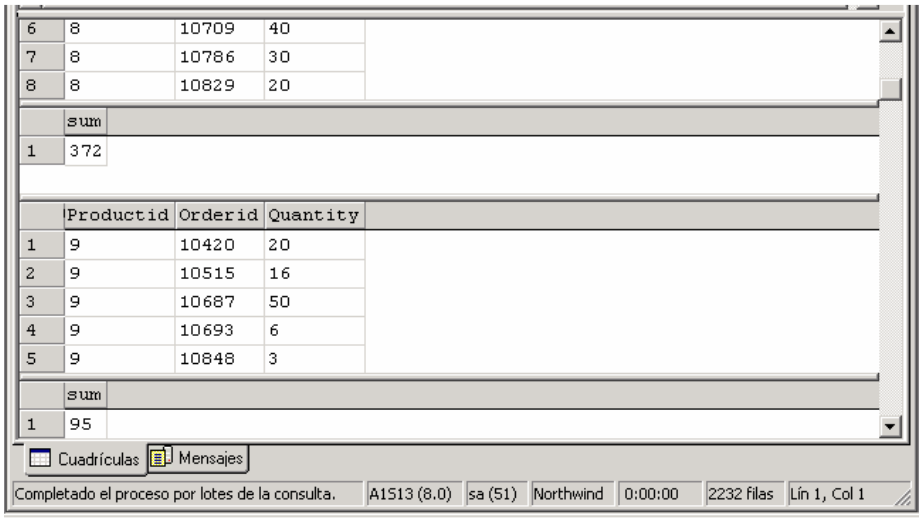
	sum
1	51317

Vemos que muestra la columna con la suma total de todas las cantidades.

COMPUTE BY

Hace un resumen similiar al realizado mediante Cube o Rollup. Realiza grupos del campo indicado, y muestra el resultado de la función indicada en una columna aparte por cada grupo que haya. Su formato es Compute Función(campo) By Campo
NOTA: Los campos por los que van después de la clausula **By** deben ir incluidos en la clausula ORDER BY y además en el mismo orden en el que aparecen.

```
Select Productid, Orderid, Quantity
From [Order Details]
order by ProductID, Orderid
compute Sum(quantity) by productId
```



	Productid	Orderid	Quantity
6	8	10709	40
7	8	10786	30
8	8	10829	20

	sum
1	372

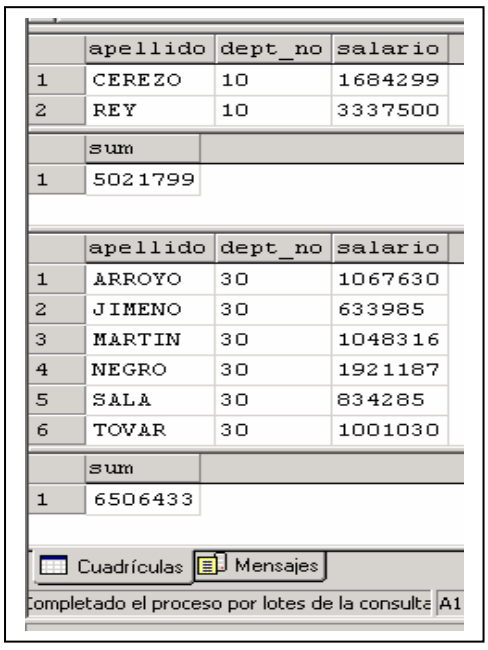
	Productid	Orderid	Quantity
1	9	10420	20
2	9	10515	16
3	9	10687	50
4	9	10693	6
5	9	10848	3

	sum
1	95

Cuadrículas | Mensajes

Completado el proceso por lotes de la consulta. | A1513 (8.0) | sa (51) | Northwind | 0:00:00 | 2232 filas | Lín 1, Col 1

```
select apellido, dept_no, salario
from emp
order by dept_no, apellido
compute sum(salario) by dept_no
```



	apellido	dept_no	salario
1	CEREZO	10	1684299
2	REY	10	3337500

	sum
1	5021799

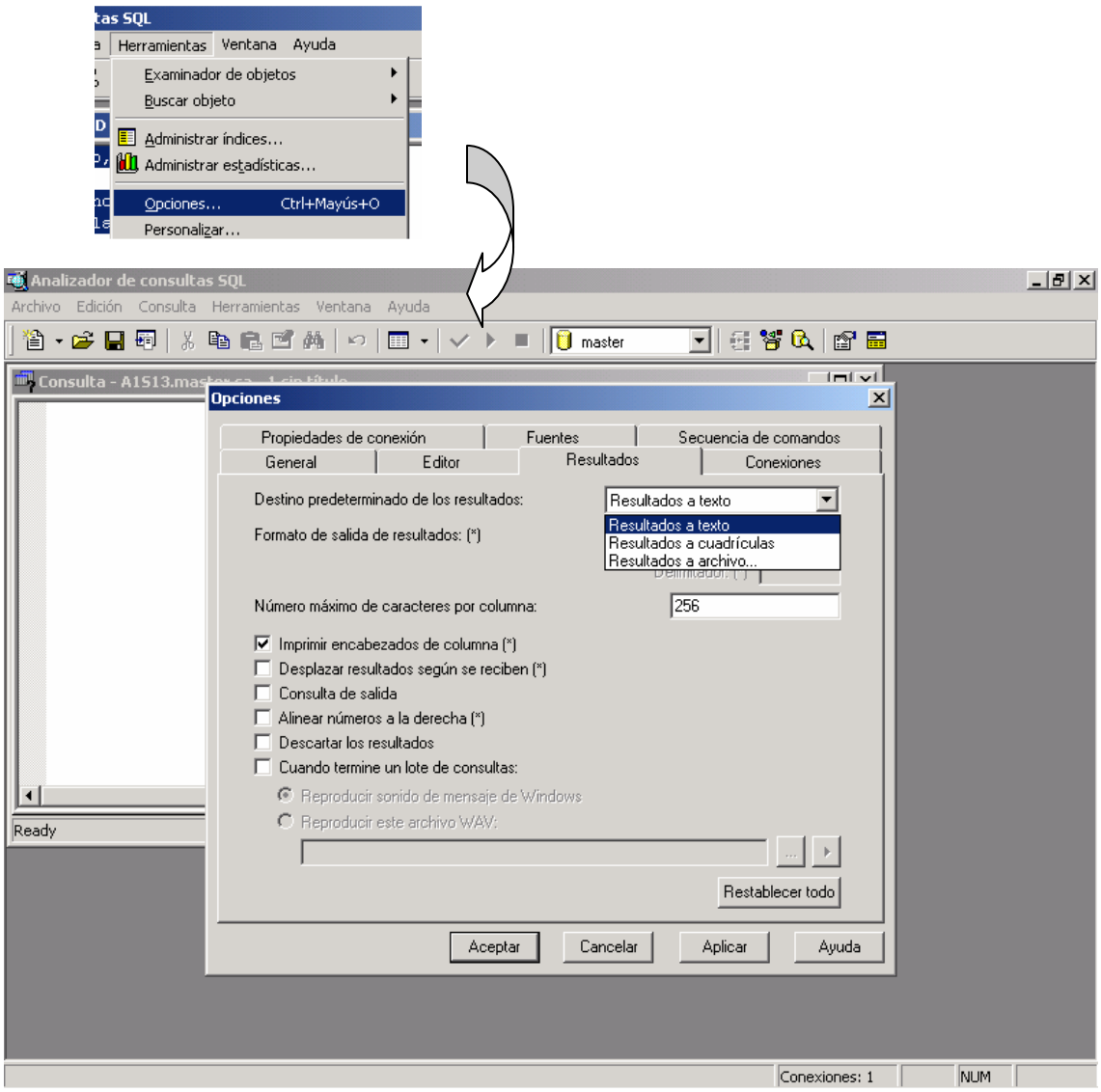
	apellido	dept_no	salario
1	ARROYO	30	1067630
2	JIMENO	30	633985
3	MARTIN	30	1048316
4	NEGRO	30	1921187
5	SALA	30	834285
6	TOVAR	30	1001030

	sum
1	6506433

Cuadrículas | Mensajes

Completado el proceso por lotes de la consulta | A1

El resultado de este procedimiento se puede mostrar en lugar de en cuadrículas, mediante texto, para ello, realizaremos los siguientes pasos:



EJERCICIOS CON COMPUTE:

1) Generar un resumen de subtotales en una consulta que devuelva el número de pedido, la cantidad pedida para todos los orderid 11070

```
Select Orderid as [Nº PEDIDO], Quantity AS [CANTIDAD]
From [Order Details]
where orderid >= 11070
order by Orderid
compute Sum(quantity) by orderid
compute sum(quantity)
```

```
Nº PEDIDO   CANTIDAD
----------- --------
11070       40
11070       20
11070       30
11070       20

            sum
            ============
            110


            sum
            ============
            110

(6 filas afectadas)
```

2) Generar un resumen con calculo de subtotales similar al anterior pero para los orderid 11075 y 11076

```
Select Orderid as [Nº PEDIDO], Quantity as [CANTIDAD]
From [Order Details]
where orderid = 11075 or orderid = 11076
order by Orderid
compute Sum(quantity) by ordered
```

```
Nº PEDIDO   CANTIDAD
----------- --------
11075       10
11075       30
11075       2

            sum
            ============
            42


Nº PEDIDO   CANTIDAD
----------- --------
11076       20
11076       20
11076       10

            sum
            ============
            50

(8 filas afectadas)
```

3) Modificación al ejercicio anterior: Agregar la cantidad total y la cantidad promedio al final del informe

```
Select Orderid as [Nº PEDIDO], Quantity as [CANTIDAD]
From [Order Details]
where orderid = 11075 or orderid = 11076
order by Orderid
compute Sum(quantity) by orderid
compute sum(quantity)
compute avg(quantity)
```

```
Nº PEDIDO   CANTIDAD
----------- --------
11075       10
11075       30
11075       2

        sum
        ===========
        42


Nº PEDIDO   CANTIDAD
----------- --------
11076       20
11076       20
11076       10

        sum
        ===========
        50


        sum
        ===========
        92


        avg
        ===========
        15

(10 filas afectadas)
```

4) Seleccionar de forma agrupada por tipo, todos los tipos, la suma de los precios y la suma del anticipo de la tabla títulos

```
Select type as [TIPO], price as [PRECIO], advance as [ANTICIPO]
From titles
order by type
compute sum(price) by type
compute avg (advance) by type
```

```
TIPO        PRECIO              ANTICIPO
------------ --------------------- ----------------------
business     19.9900             5000.0000
business     11.9500             5000.0000
business     2.9900              10125.0000
business     19.9900             5000.0000
```

```
          sum
          ======================
          54.9200


                        avg
                        ======================
                        6281.2500

TIPO        PRECIO              ANTICIPO
----------- ------------------- ---------------------
mod_cook     19.9900             .0000
mod_cook     2.9900             15000.0000

          sum
          ======================
          22.9800


          avg
          ======================
          7500.0000

TIPO        PRECIO              ANTICIPO
----------- ------------------- ---------------------
popular_comp 22.9500             7000.0000
popular_comp 20.0000             8000.0000
popular_comp NULL                NULL

          sum
          ======================
          42.9500


          avg
          ======================
          7500.0000

TIPO        PRECIO              ANTICIPO
----------- ------------------- ---------------------
psychology   21.5900             7000.0000
psychology   10.9500             2275.0000
psychology   7.0000              6000.0000
psychology   19.9900             2000.0000
psychology   7.9900              4000.0000

          sum
          ======================
          67.5200


           avg
           ======================
           4255.0000

(30 filas afectadas)
```

5) Obtener todos los tipos, precios y anticipos de la tabla titulos individualmente ordenados por el tipo en un informe que, además, muestre la suma total de los precios y anticipo por cada tipo

```
Select type as [TIPO], price as [PRECIO], advance as [ANTICIPO]
From titles
order by type
compute sum(price) by type
compute sum(advance) by type
```

```
TIPO       PRECIO              ANTICIPO
---------- ------------------- ---------------------
business    19.9900             5000.0000
business    11.9500             5000.0000
business    2.9900             10125.0000
business    19.9900             5000.0000

          sum
          ======================
          54.9200


          sum
          ======================
          25125.0000


TIPO       PRECIO              ANTICIPO
---------- ------------------- ---------------------
mod_cook    19.9900              .0000
mod_cook    2.9900             15000.0000

          sum
          ======================
          22.9800


          sum
          ======================
          15000.0000


TIPO       PRECIO              ANTICIPO
---------- ------------------- ---------------------
popular_comp 22.9500             7000.0000
popular_comp 20.0000             8000.0000
popular_comp NULL               NULL

          sum
          ======================
          42.9500


          sum
          ======================
          15000.0000


(30 filas afectadas)
```

6) Generar un informe que muestre la suma de los precios de los libros de psicología de cada editor

```
Select type AS [TIPO]
,pub_id as [CODIGO EDITOR]
,price as [PRECIO]
From titles
where type = 'psychology'
order by pub_id
compute sum(price) by pub_id
```

```
TIPO       CODIGO EDITOR PRECIO
---------- ------------- ---------------------
psychology 0736          10.9500
psychology 0736          7.0000
psychology 0736          19.9900
psychology 0736          7.9900

                  sum
                  ======================
                  45.9300

TIPO       CODIGO EDITOR PRECIO
---------- ------------- ---------------------
psychology 0877          21.5900

                  sum
                  ======================
                  21.5900

(7 filas afectadas)
```

7) Generar un informe que obtenga la suma de los precios de todos los libros de psicología, así como la suma de los precios de los libros de psicología por editor

```
Select type as [TIPO]
,pub_id as [CODIGO EDITOR]
,price as [PRECIO]
From titles
where type = 'psychology'
order by pub_id
compute sum(price) by pub_id
compute sum(price)
```

```
TIPO       CODIGO EDITOR PRECIO
---------- ------------- ---------------------
psychology 0736          10.9500
psychology 0736          7.0000
psychology 0736          19.9900
psychology 0736          7.9900

                  sum
                  ======================
                  45.9300
```

```
TIPO         CODIGO EDITOR PRECIO
------------ ------------- ---------------------
psychology   0877          21.5900

                    sum
                    ======================
                    21.5900


                    sum
                    ======================
                    67.5200

(8 filas afectadas)
```

8) Generar un informe que obtenga la suma de los precios y los anticipos para cada tipo de libro de cocina

```
Select type as [TIPO]
,pub_id [CODIGO EDITOR]
,price as [PRECIO]
,advance as [ANTICIPO]
From titles
where type like '%cook'
order by type, advance
compute sum(price) by type
compute sum(advance) by type
```

```
TIPO         CODIGO EDITOR PRECIO               ANTICIPO
------------ ------------- -------------------- --------------------
mod_cook     0877          19.9900              .0000
mod_cook     0877          2.9900               15000.0000

                    sum
                    ======================
                    22.9800


                                  sum
                                  ======================
                                  15000.0000

TIPO         CODIGO EDITOR PRECIO               ANTICIPO
------------ ------------- -------------------- --------------------
trad_cook    0877          11.9500              4000.0000
trad_cook    0877          20.9500              7000.0000
trad_cook    0877          14.9900              8000.0000

                    sum
                    ======================
                    47.8900


                                  sum
                                  ======================
                                  19000.0000

(9 filas afectadas)
```

9) Generar un informe que muestre todos los títulos, precio y anticipo de aquellos títulos que tengan un precio superior a 20 dólares. Ofrecerá también la suma total del precio y del anticipo.

```
Select title as [TITULO]
,price as [PRECIO]
,advance as [ANTICIPO]
From titles
where price > 20
order by title
compute sum(price)
compute sum(advance)
```

```
TITULO                                                                          PRECIO               ANTICIPO
------------------------------------------------------------------------------ --------------------- ---------------
-------
But Is It User Friendly?                                                         22.9500               7000.0000
Computer Phobic AND Non-Phobic Individuals: Behavior Variations                  21.5900
7000.0000
Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean                  20.9500
7000.0000

sum
======================
65.4900


sum
======================
21000.0000


(5 filas afectadas)
```

10) Generar un informe que muestre el precio, el tipo y el número de editor de la tabla titles de todos aquellas obras que pertenezcan la categoría de "business". Además aparecerá la suma total de precios y el máximo pub_id de esta selección

```
Select type as [TIPO]
,price as [PRECIO]
,pub_id as [CODIGO EDITOR]
From titles
where type = 'business'
order by pub_id
compute sum(price)
compute max(pub_id)
```

```
TIPO         PRECIO               CODIGO EDITOR
------------ -------------------- -------------
business      2.9900               0736
business     19.9900               1389
business     19.9900               1389
business     11.9500               1389

          sum
          ======================
          54.9200

          max
          ====
          1389
 (6 filas afectadas)
```

11) Realizar una consulta que recupere el tipo, precio y anticipo de la tabla titles para los libros de cocina. Mostrará la suma de los precios y los anticipos por tipo y luego, calculará el total general de los precios y los anticipos para todos los registros seleccionados.

```
Select type as [TIPO]
,price as [PRECIO]
,advance as [ANTICIPO]
From titles
where type like '%cook'
order by type
compute sum(price) by type
compute sum(advance) by type
compute sum(price)
compute sum(advance)
```

```
TIPO       PRECIO              ANTICIPO
---------- -------------------- ---------------------
mod_cook     19.9900             .0000
mod_cook     2.9900              15000.0000

          sum
          ======================
          22.9800


          sum
          ======================
          15000.0000


TIPO       PRECIO              ANTICIPO
---------- -------------------- ---------------------
trad_cook    20.9500             7000.0000
trad_cook    11.9500             4000.0000
trad_cook    14.9900             8000.0000

          sum
          ======================
          47.8900


          sum
          ======================
          19000.0000


          sum
          ======================
          70.8700


           sum
           ======================
           34000.0000

(11 filas afectadas)
```

ROLLUP

1) Realizar una consulta que resuma la cantidad de artículos pedidos por cada índice de producto y número de pedido. Realizando un cálculo acumulativo.

```
select productid as [PRODUCTO]
,orderid as [Nº PEDIDO]
,sum(quantity) as [SUMA TOTAL]
from [Order Details]
group by productid, orderid
with rollup
order by productid, orderid
```

```
PRODUCTO    Nº PEDIDO   SUMA TOTAL
----------- ----------- -----------
NULL        NULL        51317
1           NULL        828
1           10285       45
1           10294       18
1           10317       20
1           10348       15
1           10354       12
1           10370       15
1           10406       10
1           10413       24
1           10477       15
1           10522       40
1           10526       8
1           10576       10
1           10590       20
…
```

2) Modificación al ejercicio anterior: Realizar el mismo resumen pero únicamente para el producto cuyo id es 50

```
select productid as [PRODUCTO]
,orderid as [Nº PEDIDO]
,sum(quantity) as [SUMA TOTAL]
from [Order Details]
where productid = 50
group by productid, orderid
with rollup
order by productid, ordered
```

```
PRODUCTO    Nº PEDIDO   SUMA TOTAL
----------- ----------- -----------
NULL        NULL        235
50          NULL        235
50          10350       15
50          10383       15
50          10429       40
50          10465       25
50          10637       25
50          10729       40
50          10751       20
50          10920       24
50          10948       9
50          11072       22

(12 filas afectadas)
```

3) Realizar un resumen por medio de CUBE y de GROUPING sobre la modificación del ejercicio anterior de tal manera que sea posible obtener un resumen por producto y por pedido.

```
select Productid as [PRODUCTO]
,grouping(productid) as [GRUPO PRODUCTO]
,orderid as [Nº PEDIDO]
,grouping(orderid) as [GRUPO PEDIDO]
,sum(quantity) as [SUMA TOTAL]
from [Order Details]
where productid = 50
group by Productid,orderid
with cube
order by productid,orderid
```

PRODUCTO	GRUPO PRODUCTO	Nº PEDIDO	GRUPO PEDIDO	SUMA TOTAL
NULL	1	NULL	1	235
NULL	1	10350	0	15
NULL	1	10383	0	15
NULL	1	10429	0	40
NULL	1	10465	0	25
NULL	1	10637	0	25
NULL	1	10729	0	40
NULL	1	10751	0	20
NULL	1	10920	0	24
NULL	1	10948	0	9
NULL	1	11072	0	22
50	0	NULL	1	235
50	0	10350	0	15
50	0	10383	0	15
50	0	10429	0	40
50	0	10465	0	25
50	0	10637	0	25
50	0	10729	0	40
50	0	10751	0	20
50	0	10920	0	24
50	0	10948	0	9
50	0	11072	0	22

(22 filas afectadas)

¿Qué filas son de resumen? ¿Cuáles el resumen por producto y cuáles por pedido?

TRANSACCIONES

Una transacción es un conjunto de instrucciones de manipulación de datos que se ejecutan en una misma unidad de trabajo. El ejemplo más claro son las transacciones bancarias.

- Inicio de una transacción:
  BEGIN TRAN[SACTION][NombreTransaccion]
- Validación de transacción:
  COMMIT TRAN[SACTION] [NombreTransaccion]
- Decalración de punto de control:
  SAVE TRAN[SACTION] [NombrePuntoControl]
- Anulación de Transacción:

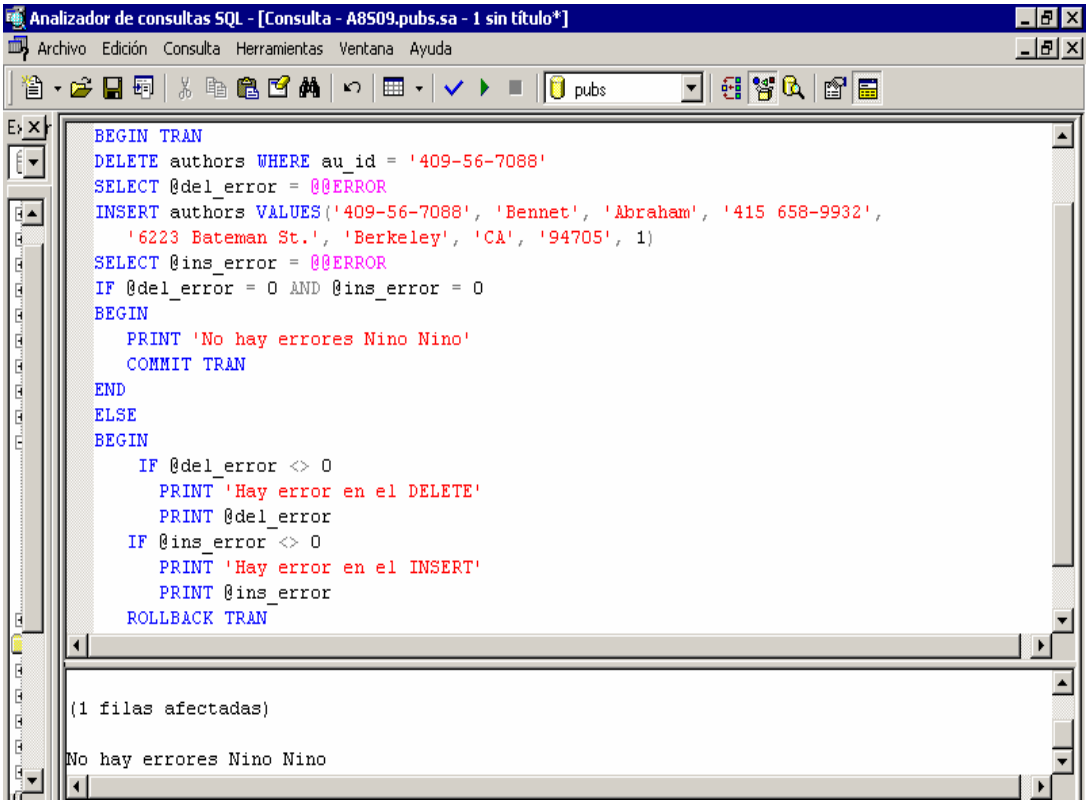
ROLLBACK TRAN[SACTION] [NombreTransaccion | NombrePuntoControl]

```
BEGIN TRAN Modificación
UPDATE Tabla SET ....
UPDATE Tabla1 SET ....
SAVE TRAN a
UPDATE Tabla2 SET ....
UPDATE Tabla3 SET ....
ROLLBACK TRAN a
ROLLBACK TRAN Modificacion
COMMIT TRAN Modificación
```

El primer Rollback guardaria las que estan en el Begin y el punto de salvamento, deshaciendo las demás ordenes.

Ejemplo de transacciones usando @@ERROR, con esto se almacena el número del error que se ha producido en última instancia. Se actualiza en cada instrucción, por eso después de cada sentencia se debe comprobar el valor de la variable para ver si tiene error. Lo que se hace es crear variables en cada instrucción almacenando el valor de @@ERROR en cada una de ellas. El valor de las variables debe ser entero.

```
USE Pubs
GO
DECLARE @del_error int, @ins_error int
BEGIN TRAN
DELETE authors WHERE au_id = '409-56-7088'
SELECT @del_error = @@ERROR

INSERT authors VALUES ('409-56-7008', 'Bennet', 'Abraham', '415 658-9932',
  '6223 Bateman St.', 'Berkeley', 'CA', '94705', 1)
SELECT @ins_error = @@ERROR

IF @del_error = 0 AND @ins_error = 0
BEGIN
  PRINT 'No hay errores Nino Nino'
  COMMIT TRAN
END
ELSE
BEGIN

  IF @del_error <> 0
    PRINT 'Hay error en el DELETE'
    PRINT @ins_error
  IF @ins_error <> 0
    PRINT 'Hay error en el INSERT'
    PRINT @ins_error
  ROLLBACK TRAN
END
GO
```



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
Hour	Hh
Minute	mi, n
Second	ss, s
Millisecond	Ms

GetDate

```
select getdate()→ Funcion que recupera la fecha actual del sistema.
```

Convert, Cast

Convierten explícitamente una expresión de un tipo de datos en otro. CAST y CONVERT proporcionan funciones similares.
Convert(Tipodedatosdestino, Origen, Estilo)

- *Tiposdedatosdestino:* Siempre ha de ser tipo carácter
- *Origen:* Puede ser tipo fecha, numérico o moneda.
- *Estilo:* Opcional. Es un código que indica el formato en el que devuelve la cadena de caracteres.

Sintaxis:

Convert(TipoDato,Dato)
Cast (Dato as TipoDato)

```
Ejemplo:

declare @n int,@palabra nvarchar(10)
set @n = 1
set @palabra ='Número'
print convert(nvarchar(2),@n) + ' ' + @palabra
print cast(@n as nvarchar(2)) + ' ' + @palabra

while (@n<11)
      begin
            print convert(nvarchar(2),@n) + ' ' + @palabra
            print cast(@n as nvarchar(2)) + ' ' + @palabra
      set @n = @n + 1
      end
```

DateName

Devuelve una cadena de caracteres que representa la parte de la fecha especificada de la fecha especificada
Los calculos para las horas no son exactos cuando se trata de SmallDateTime, por lo que devuelve 0.

```
select datename(month,fecha_alt) as 'Nombre mes' from emp where emp_no = 7867
→ diciembre
select datename(m,fecha_alt) as 'Nombre mes' from emp where emp_no =
7867 →diciembre
select datename(week,fecha_alt) as 'Numero Semana' from emp where
emp_no = 7867 →51
select datename(week,fecha_alt) as 'Numero Semana' from emp where
emp_no = 7867 → 51
select datename( weekday,fecha_alt) as 'Dia Semana' from emp where
emp_no=7867 → miercoles
select datename(dw,fecha_alt) as 'Dia Semana' from emp where
emp_no=7867 → miercoles
```

Horas:

```
select datename(mi,fecha_alt) as 'Minutos' from emp where emp_no =
7867
select datename( minute,getdate()) as 'minutos'
select datename( mi,getdate()) as 'minutos'
select datename( hh,fecha_alt) as 'hora' from emp where emp_no = 7867
select datename( hour,getdate()) as 'hora'-->17
select datename( hh,getdate()) as 'hora'-->17
```

DatePart

Devuelve la parte de la fecha u hora indicada. Sintaxis:
Datepart(Valoradevolver, fecha)

```
select datepart(mm,fecha_alt) as 'Mes',apellido from emp-->11
select datepart(hh,getdate()) as 'Hora'-->17
select datepart(mi,getdate()) as 'Minutos'-->54
```

Nombres de Fechas

```
Day(fecha)
Devuelve un INT, equivale a datepart

select day(getdate()) as dia-->12
select datepart(dd,getdate())-->12
select month(getdate())-->7
select datepart(mm,getdate())-->7
select year(getdate())-->2002
select year(fecha_alt) from emp where emp_no = 7867
select datediff(yyyy,fecha_alt,getdate()) as 'Diferencia' from emp
where emp_no = 7867
```

DateAdd

```
DateAdd( datepart , number, date )
Añade un número a la fecha puesta
DatePart es el formato de lo que queremos añadir.
Number es el número que queremos incrementar la fecha expuesta.
```

```
select convert(datetime,'1-1-02')
select dateadd(dd,7,'1-1-02')
```

DateDiff

Devuelve la diferencia entre dos fechas en el intervalo que le indiquemos. Sintaxis:

DateDiff (DatoqueDevuelve, Fecha1, Fecha2)

- *Datoquedevuelve*: Indicamos como queremos que haga la comparación y el tipo de dato que nos devolverá, años, días, minutos etc.

```
select datediff(yyyy,fecha_alt,getdate()) as 'Diferencia' from emp
where emp_no = 7867
```

FUNCIONES MATEMATICAS

ABS

Es el valor Absoluto

```
Select ABS(-4) as 'VALOR ABSOLUTO'-->4
```

CEILING

Devuelve el entero más pequeño mayor o igual que la expresión numérica dada.

```
Select CEILING(5.4) as 'CEILING'--6
```

FLOOR

Devuelve el entero más grande menor o igual que la expresión numérica dada.

```
Select FLOOR(5) as 'FLOOR'-->5
```

POWER

Devuelve el valor de la expresión indicada elevada a la potencia especificada.

```
Select POWER(3,2) as '3 ELEVADO A 2'-->9
```

RAND

Devuelve un valor float aleatorio de 0 a 1.
Las llamadas repetitivas de RAND() en una única consulta producirán el mismo valor.

```
Select RAND(6) as 'ALEATORIO'--0.71368515806921451
Select RAND(6) as 'ALEATORIO'--0.71368515806921451
Select RAND(4) as 'ALEATORIO'--0.7136478921266981
```

Rand sobre los milisegundos actuales

```
Select RAND(DATEPART(ms,GETDATE())) as 'ALEATORIO'--
>0.71443047691954253
Select RAND(999999999)-->0.68504257551273573
```

ROUND

Devuelve una expresión numérica, redondeada a la longitud o precisión especificada.
Round(Numero, Redondeo del Número)
ROUND siempre devuelve un valor. Si *length* es un valor negativo y mayor que el número de dígitos anteriores al separador decimal, ROUND devuelve 0.

```
Select ROUND(123.4567,2)-->123.4600
Select ROUND(123.4567,-2)-->100.0000
Select ROUND(123.4567,0)-->123.0000
Select ROUND(123.4567,-3)--->0
```

SIGN

Devuelve el signo positivo (+1), cero (0) o negativo (-1) de la expresión especificada.
Dice el valor negativo, positivo o neutro (0) del valor especificado

```
Select SIGN(-3)-->-1
Select SIGN(3)-->1
Select SIGN(0)-->0
```

SQUARE

Devuelve el cuadrado de la expresión especificada.

```
Select SQUARE(4) as 'Cuadrado'-->16.0
```

SQRT

Devuelve la raíz cuadrada de la expresión especificada.

```
Select SQRT(4) as [RAIZ CUADRADA]-->2.0
```

FUNCIONES DE CADENA

ASCII

Devuelve el código ASCII del carácter más a la izquierda de una expresión de caracteres.

```
Select ASCII('A')-->65
Select ASCII('a')-->97
Select ascii('aula')-->97
```

CHAR

Una función de cadena que convierte un código ASCII int en un carácter.

```
select char(65)-->A
select char(97)-->a
```

CHARINDEX

Devuelve la posición inicial de la expresión especificada en una cadena de caracteres.

```
CHARINDEX ( expression1 , expression2 [ , start_location ] )

Argumentos
expression1
Es una expresión que contiene la secuencia de caracteres que se desea
buscar.
Expression1 es una expresión del tipo de cadenas cortas de caracteres.

Expression2
Es una expresión, normalmente una columna,
en la que se busca la cadena especificada.
Expression2 es de la categoría del tipo de datos cadena de caracteres.

start_location
Es la posición del carácter de expression2 en el que se empieza la
búsqueda de expression1.
Si no se especifica start_location, es un número negativo o es cero,
la búsqueda empieza al principio de la cadena expression2.

Si expression1 no se encuentra en expression2, CHARINDEX devuelve 0.
Si alguno de los dos es null, devuelve null

select charindex('cie','murcielago')-->4
select charindex('cie','murcielago',2)-->4
select charindex('cie','murcielago',5)-->0
select charindex('cie','murcielago',-6)-->4
```

LEFT

Devuelve la parte de una cadena de caracteres que comienza en un número de caracteres especificado a partir de la izquierda

```
select left('murcielago',5)-->murci
```

RIGHT

Devuelve la parte de una cadena de caracteres que comienza en el número de caracteres especificado en integer_expression a partir de la derecha.

```
select right('hola que tal',5)-->e tal
```

LEN

Cuenta el número de caracteres que se incluyen en la cadena.

```
select len('murcielago')-->10
```

LOWER

Convierte a Minúsculas la cadena especificada

```
select lower('MurcIELaGO') as [minusculas]-->murcielago
```

UPPER

Convierte a Mayúsculas la cadena especificada

```
select upper('murcielago') as [MAYUSCULAS]-->MURCIELAGO
```

RTRIM y LTRIM

Elimina los espacios que existen a la izquierda y a la derecha respectivamente.

```
select Rtrim ('   murcielago       ') AS [SIN ESPACIOS]-->murcielago
select Ltrim ('   murcielago       ') AS [SIN ESPACIOS]-->murcielago
select ltrim(rtrim('   hola   '))+'.'
```

REPLACE

Reemplaza por una tercera expresión todas las apariciones de la segunda expresión de cadena proporcionada en la primera expresión de cadena

```
select replace('hola que tal estas','a','A')-->holA que tAl estAs
select replace('buenos dias, que tal estas','ue','ñññ')-->bñññnos
dias,qñññ tal estas
```

SPACE

Coloca el número de espacios que se le indiquen para entre una cadena de caracteres.

```
select 'hola'+space(5)+'que tal'-->hola     que tal
```

SUBSTRING

```
      Devuelve parte de una expresión de caracteres, binaria, de texto
      o de imagen.
Sintaxis:

SUBSTRING (Expresión , Comienzo , Duración )

Argumentos
expression
Es una cadena de caracteres, cadena binaria, texto, imagen,
columna o expresión que incluye una columna.
No deben usarse expresiones que incluyan funciones de agregado.

start
Es un entero que especifica el punto en que comienza la subcadena.

length
Es un entero que especifica la longitud de la subcadena
(el número de caracteres o bytes que se devuelven).

select substring('murcielago',3,5)-->rciel
select substring('murcielago',3,len('murcielago'))-->rciel
```

REVERSE

```
      Devuelve invertida una expresión de carácter.

select reverse('hola')
```

REPLICATE

```
      Repite una expresión de caracteres un número especificado de
veces.
      select replicate('murcielago',5)

replicate, replicate, replicate, replicate, replicate
```

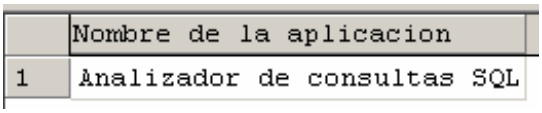
STUFF

```
      Elimina el número de caracteres especificado e inserta
      otro conjunto de caracteres en un punto de inicio indicado.

Select STUFF('Murcielago', 2, 3, 'ijklmn')→ Mijklmnielago
```

FUNCIONES DE SISTEMA

- **APP_NAME** () Devuelve el nombre de la aplicación actual

```
select app_name() as [Nombre de la aplicacion]
```



	Nombre de la aplicacion
1	Analizador de consultas SQL

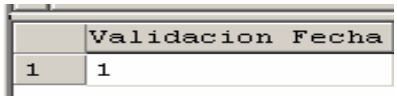
- **DATALENGTH** (Expresion) Devuelve un entero que es numero de bites necesarios para representar esa expresión → int

```
select datalength('Ninoni') as [Numero de bites]
```

	Numero de bites
1	6

- **ISDATE** (Expresion) Dice si la expresión es una fecha o no. Devuelve 1 si la expresión es una fecha y 0 si no es una fecha.

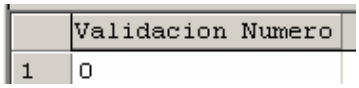
```
select isdate ('12/10/01') as [Validacion Fecha]
```



	Validacion Fecha
1	1

- **ISNUMERIC** (Expresion) Dice si la expresión es un numero o no. Devuelve 1 si la expresión es un número y 0 si la expresión no es número.

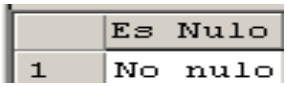
```
select isnumeric('ddd') as [Validacion Numero]
```



	Validacion Numero
1	0

- **ISNULL** (Expresión , ExpresionDevuelta) Si la expresión es nula te devuelve la Expresión Devuelta y si no devuelve la primera Expresión

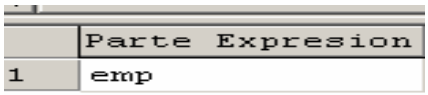
```
select isnull (null,'No nulo') as [Es Nulo]
```



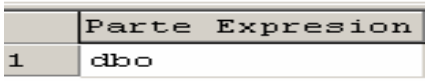
	Es Nulo
1	No nulo

- **PARSENAME** (‘NombreObjeto’, ParteNombre)
  ParteNombre es un valor de tipo Int que coge unos determinados valores del NombreObjeto:
  - 1 Objeto.
  - 2 Propietario
  - 3 Base de Datos
  - 4 Servidor

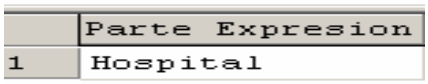
```
select parsename('Serra.Hospital.dbo.emp',1) as [Parte Expresion]
select parsename('Serra.Hospital.dbo.emp',2) as [Parte Expresion]
select parsename('Serra.Hospital.dbo.emp',3) as [Parte Expresion]
select parsename('Serra.Hospital.dbo.emp',4) as [Parte Expresion]
```



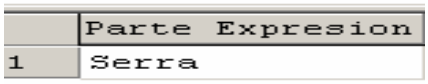
	Parte Expresion
1	emp



	Parte Expresion
1	dbo



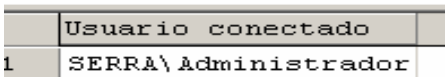
	Parte Expresion
1	Hospital



	Parte Expresion
1	Serra

- **SYSTEM_USER**
  - Devuelve el usuario del sistema actual
  - Depende de la autentificación con la que te hayas conectado

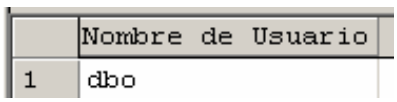
```
select system_user as [Usuario conectado]
```



	Usuario conectado
1	SERRA\Administrador

- **USER_NAME**()
  - Devuelve el nombre del usuario actual

```
select user_name()
```



	Nombre de Usuario
1	dbo

- **COALESCE** (expresión1, expresión2, ...)

  Devuelve la primera expresión no NULL
  Si todos son NULL devuelve NULL

select coalesce(Salario,Comision) from emp → Si el Salario es nulo devolverá la comisión, y si los dos son nulos, devolverá un campo null.

EJERCICIOS IF y FUNCIONES

1. Mostrar una lista de los empleados con el siguiente texto. Si el empleado no tiene nombre o la consulta devuelve null poner el texto de EMPLEADO SIN NOMBRE.

```
select isnull('El señor ' + cast(apellido as nvarchar(15))
+ ' con cargo de '
+ cast(oficio as nvarchar(15))
+ ' se dió de alta el '
+ cast(day(fecha_alt) as char(2)) + ' de '
+ cast(datename(month,fecha_alt) as nvarchar(14)) + ' de '
+ cast(year(fecha_alt) as char(4)),'EMPLEADO SIN NOMBRE') as
[DATOS EMPLEADOS]
from emp order by fecha_alt
```

```
DATOS EMPLEADOS
-----------------------------------------------------------------------------------------------
EMPLEADO SIN NOMBRE
EMPLEADO SIN NOMBRE
El señor SERRA con cargo de EMPLEADO se dió de alta el 11 de Diciembre de 1971
El señor SERRANO con cargo de DIRECTOR se dió de alta el 19 de Febrero de 1973
El señor SALA con cargo de DIRECTOR se dió de alta el 19 de Septiembre de 1976
El señor ARROYO con cargo de VENDEDOR se dió de alta el 22 de Febrero de 1981
El señor JIMENEZ con cargo de DIRECTOR se dió de alta el 2  de Abril de 1981
El señor NEGRO con cargo de DIRECTOR se dió de alta el 1  de Mayo de 1981
El señor CEREZO con cargo de DIRECTOR se dió de alta el 9  de Junio de 1981
El señor TOVAR con cargo de VENDEDOR se dió de alta el 8  de Septiembre de 1981
El señor MARTIN con cargo de VENDEDOR se dió de alta el 28 de Septiembre de 1981
El señor REY con cargo de PRESIDENTE se dió de alta el 17 de Noviembre de 1981
El señor FERNANDEZ con cargo de ANALISTA se dió de alta el 3  de Diciembre de 1981
El señor MUÑOZ con cargo de EMPLEADO se dió de alta el 23 de Junio de 1982
El señor GIL con cargo de ANALISTA se dió de alta el 30 de Marzo de 1987
El señor ALONSO con cargo de EMPLEADO se dió de alta el 3  de Mayo de 1987
El señor AGUDO con cargo de VENDEDOR se dió de alta el 19 de Noviembre de 1989
El señor MARTA con cargo de ALUMNA se dió de alta el 12 de Octubre de 2001
El señor RUIZ con cargo de ANALISTA se dió de alta el 30 de Julio de 2002

(33 filas afectadas)
```

2. Modificación del ejercicio anterior, incluir también la diferencia de años que lleva en la empresa el empleado desde la fecha actual.

```
select isnull('El señor ' + LTRIM(cast(apellido as
nvarchar(15)))
+ ' con cargo de '
+ ltrim(cast(oficio as nvarchar(15)))
+ ' se dió de alta el '
+ cast(day(fecha_alt) as char(2)) + ' de '
+ ltrim(cast(datename(month,fecha_alt) as nvarchar(15)))
+ ' de '
+ cast(year(fecha_alt) as char(4))
+ ' y lleva en la empresa '
+ ltrim(cast(datediff(yyyy,fecha_alt,getdate()) as nvarchar(5)))
+ ' años','EMPLEADO SIN NOMBRE')
 as [DATOS EMPLEADOS]
from emp order by fecha_alt
```

```
DATOS EMPLEADOS
--------------------------------------------------------------------------------
EMPLEADO SIN NOMBRE
EMPLEADO SIN NOMBRE
El señor SERRA con cargo de EMPLEADO se dió de alta el 11 de Diciembre de 1971 y
lleva en la empresa 31 años
El señor SERRANO con cargo de DIRECTOR se dió de alta el 19 de Febrero de 1973 y
lleva en la empresa 29 años
El señor SALA con cargo de DIRECTOR se dió de alta el 19 de Septiembre de 1976 y
lleva en la empresa 26 años
El señor Arias con cargo de Analista se dió de alta el 1  de Agosto de 1978 y
lleva en la empresa 24 años
El señor Toro con cargo de Director se dió de alta el 10 de Febrero de 1979 y
lleva en la empresa 23 años
El señor ARROYO con cargo de VENDEDOR se dió de alta el 22 de Febrero de 1981 y
lleva en la empresa 21 años
El señor JIMENEZ con cargo de DIRECTOR se dió de alta el 2  de Abril de 1981 y
lleva en la empresa 21 años
El señor NEGRO con cargo de DIRECTOR se dió de alta el 1  de Mayo de 1981 y lleva
en la empresa 21 años
El señor CEREZO con cargo de DIRECTOR se dió de alta el 9  de Junio de 1981 y
lleva en la empresa 21 años
El señor TOVAR con cargo de VENDEDOR se dió de alta el 8  de Septiembre de 1981 y
lleva en la empresa 21 años
El señor MARTIN con cargo de VENDEDOR se dió de alta el 28 de Septiembre de 1981
y lleva en la empresa 21 años
El señor REY con cargo de PRESIDENTE se dió de alta el 17 de Noviembre de 1981 y
lleva en la empresa 21 años
El señor FERNANDEZ con cargo de ANALISTA se dió de alta el 3  de Diciembre de
1981 y lleva en la empresa 21 años
El señor MARTA con cargo de ALUMNA se dió de alta el 12 de Octubre de 2001 y
lleva en la empresa 1 años

(33 filas afectadas)
```

3. Subir el sueldo en 5000 pts a los empleados de la plantilla del hospital La Paz en caso de que la suma de sus salarios no supere el millon de pesetas, en caso contrario bajar el sueldo en 5000 pts.

```
declare @SumaSal int
select @SumaSal = sum(salario)
from plantilla as p
inner join hospital_cod as h
on p.hospital_cod = h.hospital_cod
where h.nombre = 'La Paz'
print @sumasal
if (@SumaSal) < 1000000
      begin
--Subimos el sueldo
            update plantilla set salario = salario + 5000
            from plantilla as p
            inner join hospital_cod  as h
            on p.hospital_cod = h.hospital_cod
            where h.nombre = 'La Paz'
--Notificamos la subida
            select 'Al empleado del Hospital '
            + h.nombre
            + ', '
            + p.apellido
            + ' con función de '
            + p.funcion
            + ', se le ha subido el sueldo en 5000 pts'
            from plantilla as p
            inner join hospital_cod as h
            on p.hospital_cod = h.hospital_cod
            where h.nombre = 'La Paz'
end
```

```
else
      begin
--Bajamos el sueldo
            update plantilla set salario = salario - 5000
            from plantilla as p
            inner join hospital_cod  as h
            on p.hospital_cod = h.hospital_cod
            where h.nombre = 'La Paz'
--Notificamos la bajada
            select 'Al empleado del Hospital '
            + h.nombre
            + ', '
            + p.apellido
            + ' con función de '
            + p.funcion
            + ', se le ha reducido el sueldo en 5000 pts'
            from plantilla as p
            inner join hospital_cod as h
            on p.hospital_cod = h.hospital_cod
            where h.nombre = 'La Paz'
      end
```

Al empleado del Hospital La Paz, Higueras D. con función de Enfermera, se le ha reducido el sueldo en 5000 pts
Al empleado del Hospital La Paz, Rivera G. con función de Enfermera, se le ha reducido el sueldo en 5000 pts
Al empleado del Hospital La Paz, Carlos R. con función de Enfermera, se le ha reducido el sueldo en 5000 pts
Al empleado del Hospital La Paz, Bocina G. con función de Enfermero, se le ha reducido el sueldo en 5000 pts
Al empleado del Hospital La Paz, Núñez C. con función de Interino, se le ha reducido el sueldo en 5000 pts

(5 filas afectadas)

4. Calcular la media de años que llevan los empleados en la empresa. Si la media supera los 15 años, subir el sueldo en 20000 pts a los empleados que esten en la primera decada desde la fundación de la empresa. Si no se supera esta media, se les subirá el sueldo a los empleados que no estén en la primera decada de la empresa. Mostrar los datos después de la actualización con formato de Fecha Completa: Martes 19 Octubre 1978.

```
declare @Media int,@MinFecha smalldatetime,@Dif smalldatetime
select @MinFecha = min(fecha_alt) from emp
select @Media = avg(datediff(yy,fecha_alt,getdate())) from emp
select @dif = dateadd(yy,10,@minfecha)

if (@Media > 15)
      begin
update emp set salario = salario + 20000 where fecha_alt between
@minfecha and @dif
select datename(weekday,fecha_alt) + ' '
+ cast(datepart(day,fecha_alt) as char(2)) + ' '
+ datename(month,fecha_alt) + ' '
+ datename(year,fecha_alt) as [FECHA COMPLETA]
,Apellido
```

```
,Salario,'Actualizacion Completa' as [ACTUALIZACION] from emp
where fecha_alt between @minfecha and @dif
order by fecha_alt

      end
else
      begin
update emp set salario = salario + 10000 where fecha_alt > @dif
select datename(weekday,fecha_alt) + ' '
+ cast(datepart(day,fecha_alt) as char(2)) + ' '
+ datename(month,fecha_alt) + ' '
+ datename(year,fecha_alt) as [FECHA COMPLETA]
,Apellido
,Salario,'Actualizacion Completa' as [ACTUALIZACION]
from emp
where fecha_alt > @dif
order by fecha_alt
end
```

	FECHA COMPLETA	Apellido	Salario	ACTUALIZACION
1	Miércoles 23 Junio 1982	MUÑOZ	201066	Actualizacion Completa
2	Domingo 22 Enero 1984	ALONSO	170134	Actualizacion Completa
3	Domingo 22 Enero 1984	Alonso	88000	Actualizacion Completa
4	Lunes 30 Marzo 1987	GIL	464006	Actualizacion Completa
5	Domingo 11 Diciembre 1988	ROBERT	22000	Actualizacion Completa
6	Domingo 19 Noviembre 1989	AGUDO	25459	Actualizacion Completa
7	Viernes 12 Octubre 2001	MARTA	136400	Actualizacion Completa
8	Viernes 12 Octubre 2001	examen	6245	Actualizacion Completa
9	Jueves 7 Febrero 2002	Escriche	130000	Actualizacion Completa
10	Martes 30 Julio 2002	RUIZ	254098	Actualizacion Completa
11	Miércoles 2 Octubre 2002	Benito	179666	Actualizacion Completa
12	Lunes 14 Octubre 2002	DONTIES	66042	Actualizacion Completa

5. Mostrar los años de antigüedad de los empleados, la fecha de alta, y otro campo donde introduciremos los trienios que lleva en la empresa el trabajador hasta 7 como máximo y cuatro como mínimo. Si está fuera de este intervalo escribiremos ‘No Bonificable’.

```
Select Apellido, Fecha_Alt
,'Antigüedad' = convert(nvarchar(50)
,datediff(yy,fecha_alt, getdate())) + ' años'
,'Trienios' = case datediff(yy,fecha_alt, getdate()) / 3
When '4'
Then
     'Cuatro'
When '5'
Then
     'Cinco'
When '6'
Then
     'Seis'
When '7'
Then
     'Siete'
Else
     'No Bonificable'
End
From Emp
```

	Apellido	Fecha_Alt	Antigüedad	Trienios
10	MUÑOZ	1982-06-23 00:00:00	20 años	Seis
11	NEGRO	1981-05-01 00:00:00	21 años	Siete
12	REY	1981-11-17 00:00:00	21 años	Siete
13	ROBERT	1988-12-11 00:00:00	14 años	Cuatro
14	RUIZ	2002-07-30 00:00:00	0 años	No Bonificable
15	SALA	1976-09-19 00:00:00	26 años	No Bonificable
16	SERRA	1971-12-11 00:00:00	31 años	No Bonificable
17	SERRANO	1973-02-19 00:00:00	29 años	No Bonificable
18	TOVAR	1981-09-08 00:00:00	21 años	Siete
19	Toro	1979-02-10 00:00:00	23 años	Siete
20	Arias	1978-08-01 00:00:00	24 años	No Bonificable
21	Benito	2002-10-02 00:00:00	0 años	No Bonificable

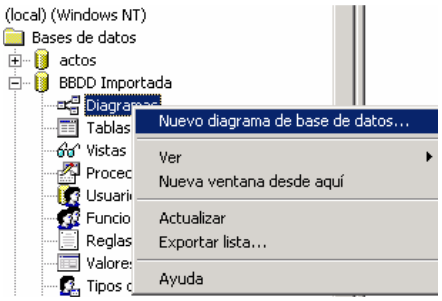
6. Calculando la suma de salarios de los directores, subir el sueldo a los que cobren entre el minimo salario y 250000 pts.
   - Mantener el sueldo a los directores si la suma está entre 1200000 y 1300000.
   - Bajar el sueldo a los que se encuentren entre 250000 y el máximo salario.
   - Mostrar los datos actualizados antes y después de los empleados que se actualicen.

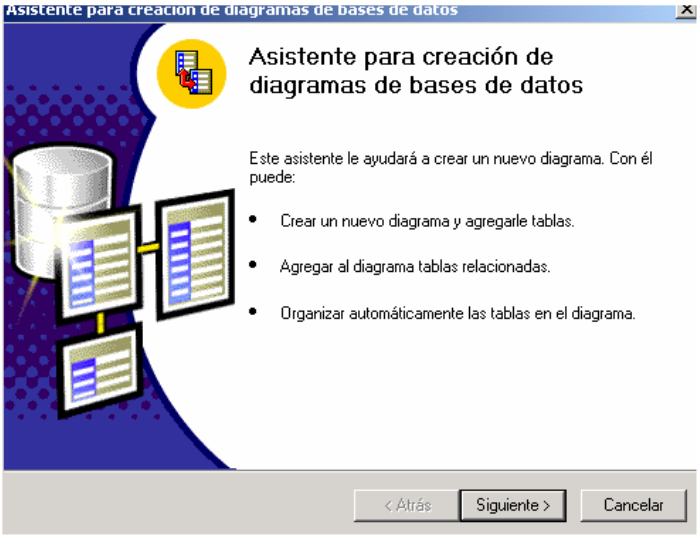
```
if (select sum(salario) from emp where oficio = 'director')
< 1200001
   BEGIN
      PRINT 'Sueldo subido a los directores'
      declare @MinSal int
      select @MinSal = min(salario) from emp where oficio =
      'director'
      print 'Antes de la Actualización'
      select top 3 with ties salario,apellido from emp where
      oficio = 'director'
      group by salario,apellido
      having salario between @MinSal and 250000
      order by salario

      update emp set salario = salario * 1.1
      where oficio = 'director'
      and
      salario between @MinSal and 250000
      update emp set salario = salario * 1.1 where oficio =
      'director'

      print 'Después de la Actualización'
      select top 3 with ties salario,apellido from emp where
      oficio = 'director'
      group by salario,apellido
      having salario between @MinSal and 250000
      order by salario
   END
else
if (select sum(salario) from emp where oficio = 'director')
between 1200001 and 1300000
begin
      print 'Mismo sueldo a los directores'
      select salario,apellido from emp where oficio = 'director'
      order by salario
      compute max(salario)
```

```
end
else
if (select sum(salario) from emp where oficio = 'director')
> 1300001
begin
      print 'Sueldo bajado a los directores'
      declare @Maxsal int
      select @Maxsal = max(salario) from emp where oficio =
      'director'
      print 'Antes de la Actualización'
      select top 3 with ties salario,apellido from emp where
      oficio = 'director'
      group by salario,apellido
      having salario between 200000 and @MaxSal
      order by salario desc
      update emp set salario = salario / 1.1
      where oficio = 'director'
      and
      salario between 200000 and @MaxSal
      print 'Después de la Actualización'
      select top 3 with ties salario,apellido from emp where
      oficio = 'director'
      group by salario,apellido
      having salario between 200000 and @MaxSal
      order by salario desc
      end

Sueldo bajado a los directores
Antes de la Actualización
salario     apellido
----------- --------------------------------------------------
418302      JIMENEZ
400729      NEGRO
344484      CEREZO

(3 filas afectadas)

(3 filas afectadas)

Después de la Actualización
salario     apellido
----------- --------------------------------------------------
380274      JIMENEZ
364299      NEGRO
313167      CEREZO

(3 filas afectadas)
```

RELACIONES ENTRE TABLAS

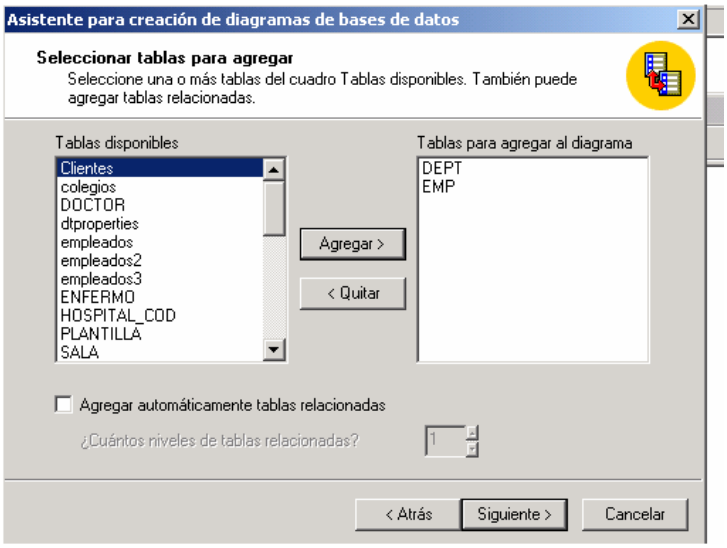
Para realizar relaciones, teniendo el foco sobre Diagramas, botón derecho -> Nuevo diagrama de base de datos.

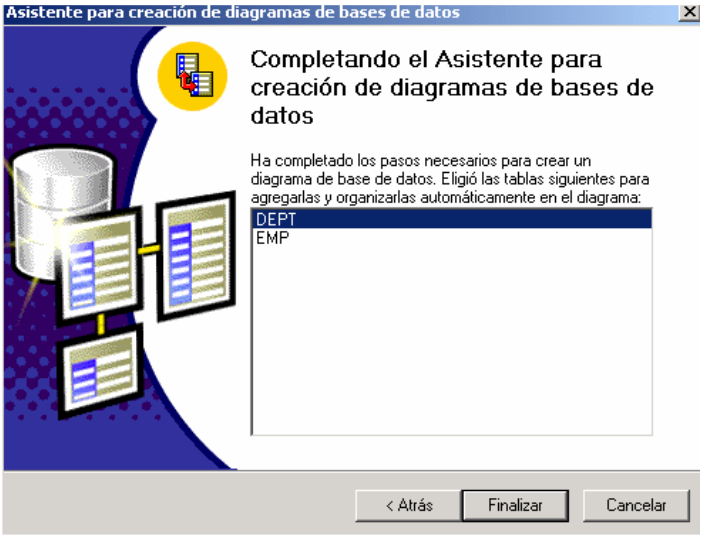




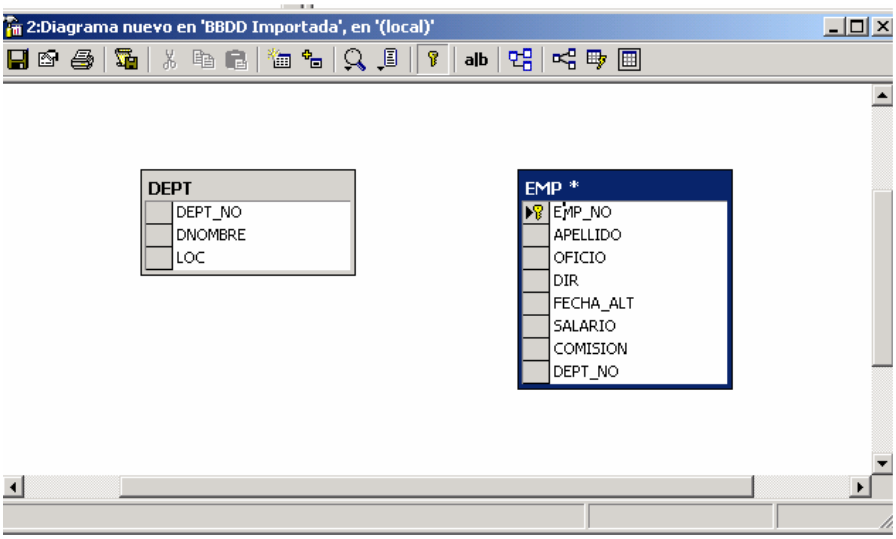
Nos saldrá un asistente para realizar los diagramas. Pulsamos siguiente

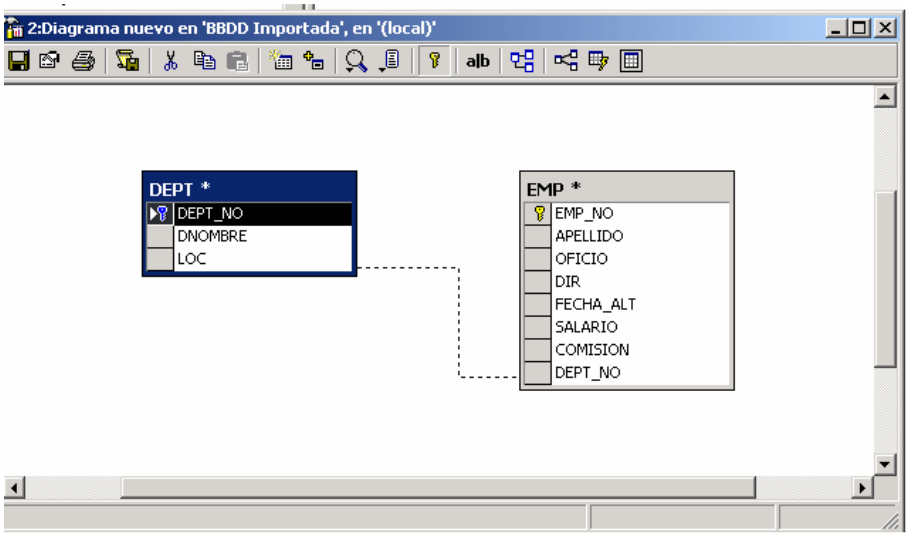
Seleccionamos las tablas que queremos relacionar...





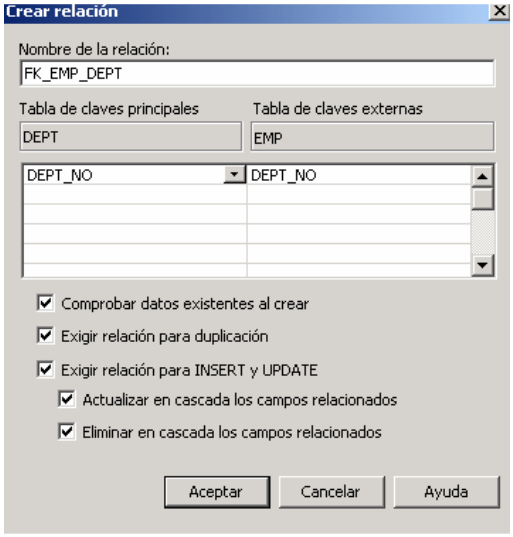
Establecemos las claves principales. Para ello, nos posicionamos sobre el campo que queramos y pulsamos el icono de la llave.

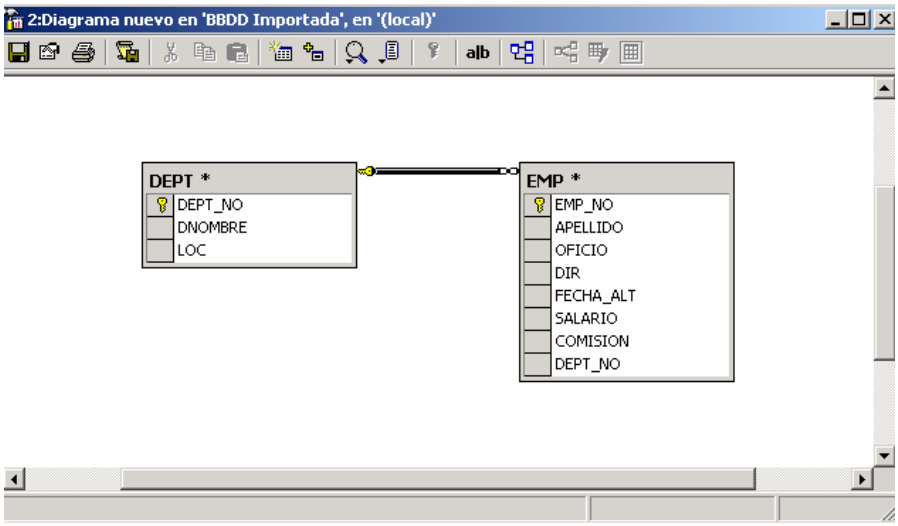




Nos posicionamos sobre el campo clave y hacemos clic sobre, el, sin soltar arrastramos hasta el campo con el que queremos establecer la relación.

Nos saldrá la pantalla de la imagen, en la que estableceremos las opciones de integridad referencial, actualización y borrado en cascada etc.





Después de pulsar aceptar, estará establecida la relación. Pulsamos el icono de guardar para guardarla.

PROCEDIMIENTOS CON PARÁMETROS

Sintaxis:

Create Procedure Procedimiento **@**NombreParametro Tipo de dato **As**
Instrucciones
Posteriormente lo ejecutamos introduciendo un valor/res correspondiente al parámetro/os introducido.
Exec Procedimiento Valor

Ejemplo 1

```
Create Procedure NumEmpDept @ndep smallint
as select count(*) as [Num Empleados], dept_no
from emp
where dept_no = @ndep
group by dept_no
```

Crea el procedimiento numempdept con el parámetro ndep que es tipo smallint (equivalente al short en visual).
El procedimiento hace una select que realiza grupos por número de deparamento y cuenta cuantos empleados hay en cada departamento.
Exec NumEmpDept 20
Ejecuta el procedimiento introduciendo el valor 20 para el parámetro ndep, con lo que haría la select sobre el departamento 20.

Ejemplo 2

```
CREATE PROCEDURE BuscaEmp @ndep smallint, @noficio nvarchar(19) AS
Select * from emp
```

PROCEDIMIENTOS ALMACENADOS CON PARÁMETROS CON VALORES POR DEFECTO

Sintaxis:
Create Procedure Nombre **@**Variable **tipo** = Valor
As
Instrucciones
Donde Valor es el valor que le damos por defecto, este valor puede almacenar comodines (como % que equivale a *).

Ejemplo 1

```
Create Procedure numemp @ndept smallint = 10
As
Select dept_no, count(*) as [num empleados]
From emp
Where dept_no =  @ndept
Group by dept_no
```

Crea un procedimiento con el parámetro ndept y le da por defecto valor 10.

Ejemplo 2

Debe devolver salario, oficio y comisión y le pasamos el apellido.

```
CREATE PROCEDURE salariooficio  @papellido varchar(20) = 'REY' AS
Select oficio, salario, comision from emp where apellido = @papellido
exec salariooficio 'Jiménez'
```

Sacaría el salario, el oficio y la comisión de todos los que tengan apellido Jiménez, sino pusiésemos parámetro, por defecto sacaría los que tuviesen apellido Rey.

Ejemplo 3

```
CREATE PROCEDURE salariooficiolike  @papellido varchar(20) = 'REY' AS

Select oficio, salario, comision from emp where apellido like '%' + @papellido + '%';
Exec salariooficio 's'
```

Sacaría oficio, salario y comisión de los empleados que tuviesen una s en su apellido.

Ejemplo 4

Introducir oficio y salario debe sacar el apellido de los empleados que tengan el mismo apellido y ganen mas del salario indicado. Debemos hacer que sino introduce nada saque todos los registros.

```
CREATE PROCEDURE dosparametros2 @oficio varchar(12) = '%',
@salario varchar(10) = '0' AS
select apellido from emp where oficio like @oficio and salario > @salario;
```

Ejemplo 5

Sacar todos los empleados que se dieron de alta entre una determinada fecha inicial y fecha final y que pertenecen a un determinado departamento.

```
CREATE PROCEDURE tresparametros
@finicial datetime = '01-01-1980',
@ffinal smalldatetime = '12-07-2002' ,
@dept_no nvarchar(10) = '%' AS
select * from emp where fecha_alt between @finicial and @ffinal
and dept_no = @dept_no
```

Ejemplo 6

Crear procedimiento que inserte un empleado. Crear otro procedimiento que borre un empleado que coincida con los parámetros indicados (los parámetros serán todos los campos de la tabla empleado)

```
CREATE PROCEDURE [Inserta Empleado]
@emp_no int,
@apellido nvarchar(20),
@oficio nvarchar(20),
@dir int,
@fecha_alt smalldatetime,
@salario int,
@comision int,
@dept_no int
AS
insert into emp values
(@emp_no,
@apellido,
@oficio,
@dir,
@fecha_alt,
@salario,
@comision,
@dept_no)

CREATE PROCEDURE [Borra Empleado]
@emp_no int,
@apellido nvarchar(20),
@oficio nvarchar(20),
@dir int,
@fecha_alt smalldatetime,
@salario int,
@comision int,
@dept_no int
AS
delete from emp where emp_no = @emp_no
and apellido = @apellido
and oficio = @oficio
and dir = @dir
and fecha_alt = @fecha_alt
and salario = @salario
and comision = @comision
and dept_no = @dept_no
```

DECLARACIÓN DE VARIABLES EN PROCEDIMIENTOS ALMACENADOS

Sintaxis de declaración
Declare @nombre tipo_dato
Sintaxis de asignación directa
Set @nombre = valor
Sintaxis de asignación mediante consulta
Select @nombre = campo from tabla

VARIABLES DE SALIDA EN PROCEDIMIENTOS ALMACENADOS

Sintaxis
Create Procedure Nombre @Variable tipodedato **Output**
Instrucciones
Print @Variable

Ejemplo 1:

```
Create Procedure TotalSalCom @Apellido nvarchar(25), @total int Output
--Creamos el procedimiento con un parámetro y una variable de salida
As
--Declaramos dos variables para almacenar valores
Declare @Sal Int
Declare @Com Int
--Asignamos los valores correspondientes a las variables y después las sumamos y las guardamos en la variable de salida.
Select @Sal = Salario from emp where Apellido = @Apellido
Select @Com = Comision from emp where Apellido = @Apellido
Set @Total = @Sal + @Com
--Devolvemos el valor de la variable que queremos
Print @total
```

Y para ejecutarlo en el analizador de consultas, hemos de declarar la variable que vamos a recuperar también, para almacenar el valor que nos pasa el procedimiento almacenado en ella. Es decir, en este caso @total almacenará el valor pasado por la variable del mismo nombre que hay en el procedimiento:

```
declare @total int
exec TotalSalCom 'Jimenez', @total output
```

Ejemplo 2:

```
Create Procedure Totales
@ndept int = null, @total int Output
As
If @ndept is null
        Select @total = Sum(Salario) from Emp
Else
        Select @total = Sum(Salario) from Emp where Dept_no = @ndept
```

EJEMPLOS:

```
CREATE PROCEDURE Dosparametros @oficio varchar(12) = '%',
@salario int = 0 AS
if @salario = 0
print 'No ha introducido salario'
else
select apellido from emp where oficio like @oficio and salario > @salario

CREATE PROCEDURE BorrarEmp @apellido varchar(12) = '0000'
,@ape nvarchar(30) = '0'
,@Num int = 0
as
if @apellido = '0000'
print 'Imposible borrar si no introduce un apellido'
else
begin
        select @ape = apellido from emp where apellido = @apellido
        select @num = count(apellido) from emp where apellido = @ape
        if (@ape = '0')
        begin
                print 'Apellido no encontrado, introduzca un apellido de la base de datos'
        end
        else
        begin
                insert into control values
                (@ape
                ,user_name()
                ,getdate()
                ,@Num)
                print 'El empleado ' + @ape + ' ha sido borrado de la base de datos'
                delete from emp where apellido = @ape

        end
end

drop procedure borraremp
```

PROCEDIMIENTOS ALMACENADOS

1) Sacar todos los empleados que se dieron de alta entre una determinada fecha inicial y fecha final y que pertenecen a un determinado departamento.

```
CREATE PROCEDURE FechasDept
@finicial datetime
@ffinal smalldatetime ,
@dept_no nvarchar(10)  AS
select * from emp where fecha_alt between @finicial and @ffinal
and dept_no = @dept_no
```

2) Crear procedimiento que inserte un empleado.

```
CREATE PROCEDURE [Inserta Empleado]
@emp_no int,
@apellido nvarchar(20),
@oficio nvarchar(20),
@dir int,
@fecha_alt smalldatetime,
@salario int,
@comision int,
@dept_no int
AS
insert into emp values
(@emp_no,
@apellido,
@oficio,
@dir,
@fecha_alt,
@salario,
@comision,
@dept_no)
```

3) Crear un procedimiento que recupere el nombre, número y número de personas a partir del número de departamento.

```
Create Procedure NumEmpDept @ndep smallint
as select count(*) as [Num Empleados], dept_no
from emp
where dept_no = @ndep
group by dept_no
```

4) Crear un procedimiento igual que el anterior, pero que recupere también las personas que trabajan en dicho departamento, pasándole como parámetro el nombre.

```
Create Procedure PersonasDept @dept nvarchar(30)
As
    Select e.dept_no as [NUMERO DEPT]
    , D.DNOMBRE as [NOMBRE], count(*) as [Numero Empleados]
    From emp as e
    inner join dept as d
    on e.dept_no = d.dept_no
    Where d.dnombre =  @dept
    Group by d.dnombre, e.dept_no

    select e.dept_no as [Nº DEPARTAMENTO]
    , d.dnombre as [DEPARTAMENTO]
    , e.APELLIDO, e.SALARIO
    from emp as e
    inner join dept as d
    on e.dept_no = d.dept_no
    where d.dnombre = @dept
```

5) Crear procedimiento para devolver salario, oficio y comisión, pasándole el apellido.

```
CREATE PROCEDURE SalarioOficio  @papellido nvarchar(20)  AS
Select oficio, salario, comision from emp where apellido = @papellido
exec SalarioOficio 'Jiménez'
```

6) Igual que el anterior, pero si no le pasamos ningún valor, mostrará los datos de todos los empleados.

```
CREATE PROCEDURE SalarioOficio  @papellido nvarchar(20) ='%' AS
Select oficio, salario, comision from emp where apellido like @papellido
exec SalarioOficio 'Jiménez'
```

7) Crear un procedimiento para mostrar el salario, oficio, apellido y nombre del departamento de todos los empleados que contengan en su apellido el valor que le pasemos como parámetro.

```
CREATE PROCEDURE salariooficiolike  @papellido varchar(20) = 'REY' AS

Select oficio, salario, comision from emp where apellido like '%' + @papellido +
'%';
Exec salariooficio 's'
```

8) Crear un procedimiento que recupere el número departamento, el nombre y número de empleados, dándole como valor el nombre del departamento, si el nombre introducido no es válido, mostraremos un mensaje informativo comunicándolo.

```
Create Procedure Departamento @dept nvarchar(30)
As
declare @DeptDef nvarchar(30)
set @DeptDef = null
select @DeptDef = dnombre from dept
where dnombre = @dept
if (@DeptDef  is  null)
    print 'Departamento introducido no valido: '  + @dept
else
begin
    Select e.dept_no as [NUMERO DEPT]
    , D.DNOMBRE as [NOMBRE], count(*) as [Numero Empleados]
    From emp as e
    inner join dept as d
    on e.dept_no = d.dept_no
    Where d.dnombre =  @dept
    Group by d.dnombre, e.dept_no
End
```

9) Crear un procedimiento para devolver un informe sobre los empleados de la plantilla de un determinado hospital, sala, turno o función. El informe mostrará número de empleados, media, suma y un informe personalizado de cada uno que muestre número de empleado, apellido y salario.

```
create procedure EmpPlantilla
@Valor nvarchar(30)
as
declare @Consulta nvarchar(30)
select @Consulta = Nombre
from hospital
where Nombre = @Valor
if (@Consulta is null)
Begin
   select @Consulta = Nombre from sala
   where Nombre = @Valor
   if (@Consulta is null)
   Begin
         select @Consulta = T from plantilla
         where T = @Valor
         if (@Consulta is null)
         Begin
               Select @Consulta = Funcion from plantilla
               where funcion = @Valor
               if (@Consulta is null)
               Begin
                        print 'El valor introducido no es un
                        hospital, Sala, Turno o Función: ' + @Valor
                  print 'Verifique los datos introducidos'
```

```
                End
                else
                Begin
                      print 'FUNCION'
                      select Funcion as [Turno]
                      ,avg(salario) as [Media]
                      ,count(empleado_no) as [Nº Empleados]
                      ,sum(salario) as [Suma] from plantilla
                      group by Funcion
                      having Funcion = @Valor

                      select empleado_no as [Nº Empleado]
                      ,Apellido, Salario, Funcion
                      from plantilla
                      where Funcion = @Valor
                End
          End
          Else
          Begin
                print 'TURNO'
                select T as [Turno]
                ,avg(salario) as [Media]
                ,count(empleado_no) as [Nº Empleados]
                ,sum(salario) as [Suma] from plantilla
                group by T
                having t = @Valor

                select T as [Turno]
                ,empleado_no as [Nº Empleado]
                ,Apellido, Salario
                from plantilla
                where T = @Valor
          End
    End
    Else
    Begin
          print 'SALA'
          select s.nombre as [SALA]
          ,avg(p.salario) as [Media]
          ,count(p.empleado_no) as [Nº Empleados]
          ,sum(p.salario) as [Suma] from plantilla as p
          inner join sala as s
          on s.hospital_cod = p.hospital_cod
          group by s.nombre
          having s.nombre = @Valor


          select s.nombre as [SALA]
          ,p.empleado_no as [Nº Empleado]
          ,p.Apellido, p.Salario
          from plantilla as p
          inner join sala as s
          on s.hospital_cod = p.hospital_cod
          where s.nombre =  @Valor
    End
End
else
Begin
    print 'HOSPITAL'
    select h.nombre as [HOSPITAL]
    ,avg(p.salario) as [Media]
```

```
   ,count(p.empleado_no) as [Nº Empleados]
   ,sum(p.salario) as [Suma] from plantilla as p
   inner join hospital as h
   on h.hospital_cod = p.hospital_cod
   group by h.nombre
   having h.nombre = @Valor

   select h.nombre as [HOSPITAL]
   ,p.empleado_no as [Nº Empleado]
   ,p.Apellido, p.salario
   from plantilla as p
   inner join hospital as h
   on h.hospital_cod = p.hospital_cod
   where h.nombre = @Valor
   End
```

10) Crear un procedimiento en el que pasaremos como parámetro el Apellido de un empleado. El procedimiento devolverá los subordinados del empleado escrito, si el empleado no existe en la base de datos, informaremos de ello, si el empleado no tiene subordinados, lo informaremos con un mensaje y mostraremos su jefe. Mostrar el número de empleado, Apellido, Oficio y Departamento de los subordinados.

```
create procedure Jefes
@Ape nvarchar(30)
AS
declare @Emp int, @Jefe int, @Sub int
select @Emp = Emp_no from emp
where apellido = @Ape
if (@Emp is null)
Begin
   print 'No existe ningun empleado con este apellido: ' + @Ape
End
else
Begin
select @Jefe = a.emp_no
,@Sub = b.emp_no
from emp as a inner join emp as b
on a.emp_no = b.dir
where b.dir = @Emp
order by b.dir
if (@Jefe is null)
Begin
   select a.emp_no as [Nº de Empleado]
   ,a.apellido as [Jefe], a.Oficio
   ,a.dept_no as [Nº Departamento]
   ,b.emp_no as [Nº Empleado]
   ,b.apellido as [Subordinado]
   ,b.Oficio
   ,b.Dept_no as [Nº Departamento]
   from emp as a
   inner join emp as b
   on b.dir = a.emp_no
   where b.emp_no = @Emp
   order by b.dir
End
Else
Begin
   select a.emp_no as [Nº de Empleado]
   ,a.apellido as [Jefe], a.Oficio
```

```
   ,a.dept_no as [Nº Departamento]
   ,b.emp_no as [Nº Empleado]
   ,b.apellido as [Subordinado]
   ,b.Oficio
   ,b.Dept_no as [Nº Departamento]
   from emp as a inner join emp as b
   on a.emp_no = b.dir
   where b.dir = @Emp
   order by b.dir
End
End
```

11) Crear procedimiento que borre un empleado que coincida con los parámetros indicados (los parámetros serán todos los campos de la tabla empleado).

```
CREATE PROCEDURE [Borra Empleado]
@emp_no int,
@apellido nvarchar(20),
@oficio nvarchar(20),
@dir int,
@fecha_alt smalldatetime,
@salario int,
@comision int,
@dept_no int
AS
delete from emp where emp_no = @emp_no
and apellido = @apellido
and oficio = @oficio
and dir = @dir
and fecha_alt = @fecha_alt
and salario = @salario
and comision = @comision
and dept_no = @dept_no
```

12) Modificación del ejercicio anterior, si no se introducen datos correctamente, informar de ello con un mensaje y no realizar la baja. Si el empleado introducido no existe en la base de datos, deberemos informarlo con un mensaje que devuelva el nombre y número de empleado del empleado introducido. Si el empleado existe, pero los datos para eliminarlo son incorrectos, informaremos mostrando los datos reales del empleado junto con los datos introducidos por el usuario, para que se vea el fallo.

```
CREATE PROCEDURE [Borra Empleado]
@emp_no int
,@apellido nvarchar(20)
,@oficio nvarchar(20)
,@dir int
,@fecha_alt smalldatetime
,@salario int
,@comision int
,@dept_no int
AS
DECLARE @VALOR nvarchar(30)
SELECT @VALOR = EMP_NO
```

```
FROM EMP where emp_no = @emp_no
and apellido = @apellido
and oficio = @oficio
and dir = @dir
and fecha_alt = @fecha_alt
and salario = @salario
and comision = @comision
and dept_no = @dept_no

IF (@VALOR IS NULL)
BEGIN
   SELECT @VALOR = EMP_NO FROM EMP WHERE EMP_NO = @EMP_NO
   IF (@VALOR IS NULL)
   BEGIN
         PRINT 'EMPLEADO NO EXISTENTE EN LA BASE DE DATOS, VERIFIQUE LOS
DATOS DEL SR ' + @APELLIDO
   END
   ELSE
   BEGIN
         PRINT 'DATOS INTRODUCIDOS ERRONEAMENTE: '
         PRINT CAST(@EMP_NO AS NVARCHAR(4)) + ' ' + @APELLIDO + ' '
         + @OFICIO + ' ' + CAST(@DIR AS NVARCHAR(4)) + ' '
         + CAST(@FECHA_ALT AS NVARCHAR(12)) + ' '
         + CAST(@SALARIO AS NVARCHAR(10)) + ' '
         + CAST(@COMISION AS NVARCHAR(10)) + ' '
         + CAST(@DEPT_NO AS NVARCHAR(4))
         SELECT @EMP_NO = EMP_NO, @APELLIDO = APELLIDO
         ,@OFICIO = OFICIO, @DIR = DIR
         ,@FECHA_ALT = FECHA_ALT, @SALARIO = SALARIO
         ,@COMISION = COMISION, @DEPT_NO = DEPT_NO
         FROM EMP where emp_no = @VALOR
         PRINT 'DATOS REALES DEL EMPLEADO: '
         PRINT CAST(@EMP_NO AS NVARCHAR(4)) + ' ' + @APELLIDO + ' '
         + @OFICIO + ' ' + CAST(@DIR AS NVARCHAR(4)) + ' '
         + CAST(@FECHA_ALT AS NVARCHAR(12)) + ' '
         + CAST(@SALARIO AS NVARCHAR(10)) + ' '
         + CAST(@COMISION AS NVARCHAR(10)) + ' '
         + CAST(@DEPT_NO AS NVARCHAR(4))
   END
END
ELSE
BEGIN
   delete from emp where emp_no = @emp_no
   and apellido = @apellido
   and oficio = @oficio
   and dir = @dir
   and fecha_alt = @fecha_alt
   and salario = @salario
   and comision = @comision
   and dept_no = @dept_no
END
```

13) Crear un procedimiento para insertar un empleado de la plantilla del Hospital. Para poder insertar el empleado realizaremos restricciones:

- **No podrá estar repetido el número de empleado.**
- **Para insertar, lo haremos por el nombre del hospital y por el nombre de sala, si no existe la sala o el hospital, no insertaremos y lo informaremos.**
- **El oficio para insertar deberá estar entre los que hay en la base de datos, al igual que el Turno.**
- **El salario no superará las 500.000 ptas.**
- **(Opcional) Podremos insertar por el código del hospital o sala y por su nombre.**

```
Create Procedure InsertarPlantilla
@Hospital nvarchar(30)
,@Sala nvarchar(30)
,@Empleado int
,@Apellido nvarchar(30)
,@Funcion nvarchar(30)
,@Turno nvarchar(2)
,@salario int
AS
declare @Hospital2 int
declare @Sala2 int
declare @Empleado2 int
declare @Apellido2 nvarchar(30)
declare @Funcion2 nvarchar(30)
declare @Turno2 nvarchar(2)

Select @Empleado2 = Empleado_no from plantilla
where Empleado_no = @Empleado
if (@Empleado2 is null)
Begin
      Select @Sala2 = Sala_cod from sala
      where nombre = @Sala
      Select @Hospital2 = Hospital_cod from hospital
      where nombre = @Hospital
      if (@Sala2 is null or @Hospital2 is null)
      Begin
            Print 'La Sala o el Hospital introducido no están en los
            datos:'
            Print 'Sala: ' + @Sala
            Print 'Hospital: ' + @Hospital
            Print ''
            Print 'No se ha realizado la inserción'
      End
      Else
      Begin
            Select @Funcion2 = Funcion from plantilla
            where funcion = @Funcion
            Select @Turno2 = t from plantilla
            where t = @Turno
            if (@Funcion2 is null or @Turno2 is null)
            Begin
                  Print 'La Función o el Turno introducidos no son
                  válidos'
```

```
                  Print 'Funcion: ' + @Funcion
                  Print 'Turno: ' + @Turno
                  Print ''
                  Print 'No se ha realizado la inserción'
            End
            Else
            Begin
                  if (@Salario > 500000)
                  Begin
                        Print 'El salario máximo para los trabajadores
                        de la plantilla es de 500.000 ptas'
                        Print 'No se ha realizado la inserción'
                  End
                  Else
                  Begin
                        Insert Into Plantilla
                        values (@Hospital2, @Sala2, @Empleado
                        , @Apellido, @Funcion, @Turno, @Salario)
                        Print 'Inserción Realizada Correctamente'
                  End
            End
      End
End
Else
Begin
      Print 'El Número de empleado introducido está repetido: ' +
      Cast(@Empleado as nvarchar(4))
      Print 'No se ha realizado la inserción'
End
```

TRIGGERS (DESENCADENADORES, DISPARADORES)

Son procedimientos asociados a una tabla. Se activan cuando se produce una eliminación, modificación o inserción en la tabla asociada.
Ejemplos de uso de un trigger

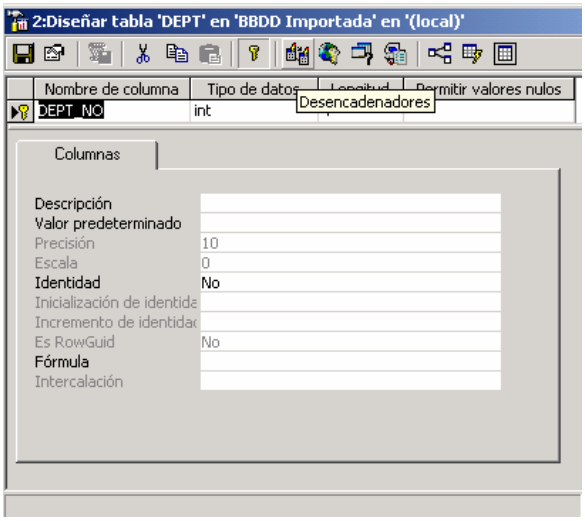
Si tenemos una tabla de clientes, y otra de informes, por ej. cada vez que se produzca una baja en clientes, almacenamos el nombre y apellidos del cliente en informe para posteriormente hacer una estadística con los clientes borrados.

Tenemos la tabla de nominas, cada vez que un usuario acceda a ella, el trigger se activa y almacena su nombre en otra tabla.

Eliminación en Cascada

Vamos a ver un ejemplo de creación de un trigger para cuando borremos un departamento se actualice la tabla empleados borrando todos aquellos cuyo departamento sea el borrado.

Entramos en el diseño de la tabla dept...

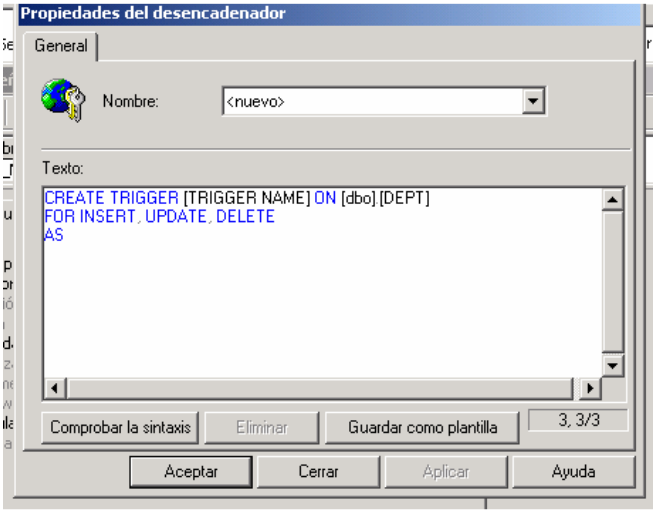


...y pulsamos el icono de desencadenadores.

Vemos que nos sale la pantalla con la sintaxis del trigger

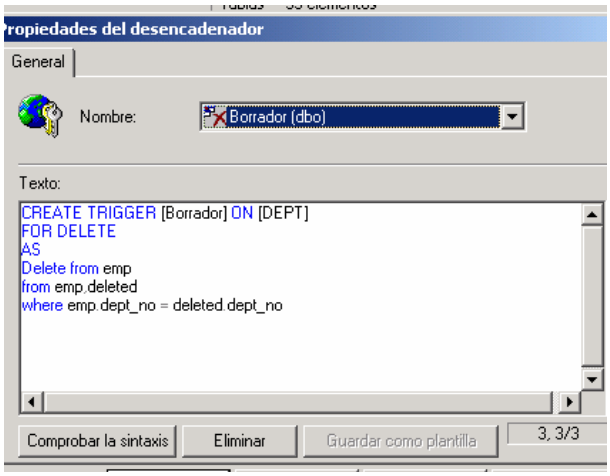
Create Trigger Nombre
On Tabla
For Delete /Insert /Update
As
Instrucciones

Como se ve en la sintaxis, le indicamos que es lo que va a hacer en el trigger detrás del For.



Ejemplo de Trigger para Delete

En Tabla en este caso como queremos borrar, obtendremos la información de la tabla Deleted, que es una tabla de sistema donde se almacenan todos los datos borrados de todas las tablas. Con lo que la instrucción quedaría así.

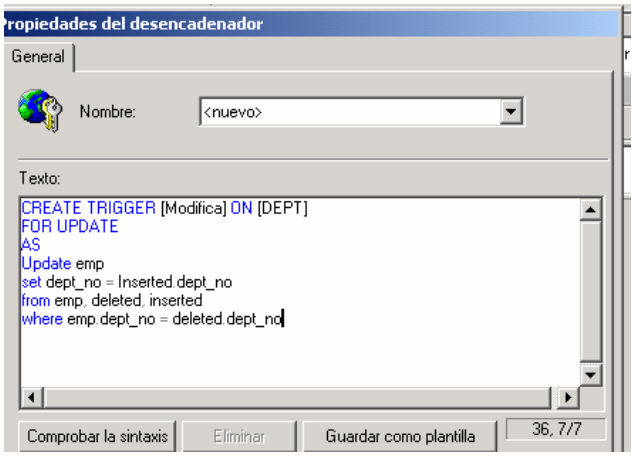



```
Create Trigger [Borrador]
on Dept
For Delete
As
delete from emp
From emp, deleted
where
emp.dept_no = deleted.dept_no
```

Con lo que si por ej. borramos el departamento 20 de la tabla dept, este departamento se almacenará en la tabla Deleted, una vez borrado, se desencadena el trigger, y borraría de la tabla empleados, todos los empleados cuyo nº de departamento, coincida con el que se ha borrado y almacenado en la tabla deleted. Si borrasemos 4 departamentos con una misma orden, se activaría el trigger con cada departamento borrado.

Ejemplo de Trigger para Update

Como hemos visto en el ejemplo anterior, para borrar existe la tabla de sistema deleted donde almacena los registros borrados, al igual que para insertar almacena los registros insertados en la tabla inserted, pero para modificar no existe ninguna tabla update, lo que hace en realidad es insertar en la tabla deleted el registro antes de ser modificado y en la tabla inserted el registro ya modificado, porque entiende que se ha borrado un registro y se ha insertado otro.
Con lo cuál para crear un trigger que se active con un update, trabajaremos con las tablas deleted e inserted.
Ejemplo Update:

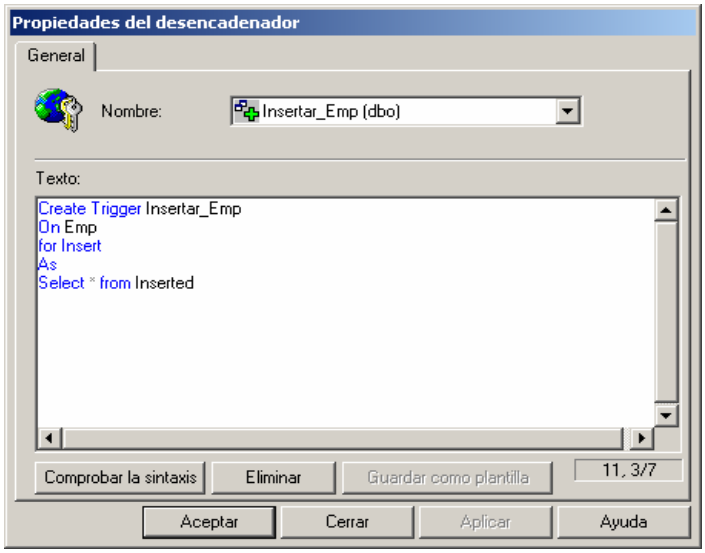



```
Create Trigger [Modificar]
On Dept
For Update
As
Update emp
set
dept_no = inserted.dept_no
from emp,deleted,inserted
where
emp.dept_no = deleted.dept_no
```

Con lo que este ejemplo lo que haría es que cuando modificamos un registro en la tabla dept, se activa el trigger, va a la tabla insert y busca los registros cuyo nº de depto. Coincida con

Ejemplo de Trigger para Insert

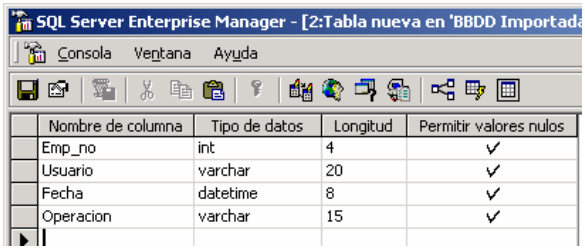
En este ejemplo inserto un nuevo empleado y lo que hago es mostrar el empleado insertado desde la tabla inserted.




```
Create Trigger
Insertar_Emp
On Emp
for Insert
As
Select * from Inserted
```

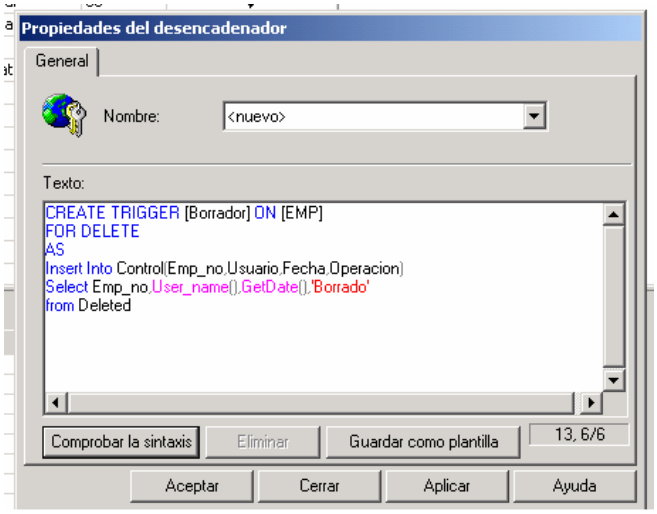
Super Ejemplo

El siguiente ejemplo almacenará en una tabla que crearemos un registro con datos de cada fila que borremos, modifiquemos o insertemos.
Este registro contendrá
Nº de empleado.
Usuario que realizó la consulta de acción.
Fecha de la consulta de acción.
Tipo de operación realizada.



1 Creamos la tabla

2 Creamos el trigger de borrado en la tabla de empleados.




```
Create Trigger Borrador on emp
For Delete
As
Insert into Control (Emp_no, Usuario, Fecha, Operacion)
Select Emp_no,User_Name(), GetDate(),'Borrado'
From Deleted
```

TRIGGERS O DESENCADENADORES

1) Crear un Trigger que borre en cascada sobre la tabla relacionada cuando borremos una sala. Mostrar el registro borrado al ejecutar el Trigger.

```
Create Trigger BorrarSala
on Sala
for delete
as
delete from plantilla from sala, deleted
where sala.sala_cod = deleted.sala_cod
select * from deleted
```

2) Crear un Trigger que se active cuando Actualicemos alguna sala del hospital, modificando sus tablas relacionadas. Mostrar el registro Actualizado.

```
Create Trigger ModificarSala
on Sala
for update
as
update plantilla
set sala_cod = inserted.sala_cod
from plantilla
where sala.sala_cod = deleted.sala_cod
select * from inserted
```

3) Crear un Trigger que se active al eliminar un registro en la tabla hospital y modifique las tablas correspondientes.

```
Create trigger BorrarHospital on hospital
for delete
as
delete from plantilla from plantilla,deleted where plantilla.hospital_cod =
deleted.hospital_cod
delete from sala from sala,deleted where sala.hospital_cod =
deleted.hospital_cod
delete from doctor from doctor,deleted where doctor.hospital_cod =
deleted.hospital_cod
```

4) Crear un Trigger para controlar la inserción de empleados, cuando insertemos un empleado se copiarán datos sobre la inserción en una tabla llamada CreateTrigger. Los datos que se copiarán son el Número de empleado, El usuario que está realizando la operación, la fecha y el tipo de operación.

```
Create Trigger [DAR ALTA]
on Emp
for insert
as
Insert into ControlTrigger (N_emp, Usuario, Fecha, Operación)
Select inserted.emp_no, user_name(), GetDate(), 'INSERCION'
```

```
From Inserted
```

5) Crear un Trigger que actue cuando se modifique la tabla hospital y sobre todas las tablas con las que esté relacionadas.

```
Create trigger Hospital on hospital
for update
as
update plantilla
set hospital_cod = inserted.hospital_cod
from plantilla, inserted, deleted
where plantilla.hospital_cod = deleted.hospital_cod
update sala
set hospital_cod = inserted.hospital_cod
from plantilla, inserted, deleted
where sala.hospital_cod = deleted.hospital_cod
update doctor
set hospital_cod = inserted.hospital_cod
from doctor, inserted, deleted
where doctor.hospital_cod = deleted.hospital_cod
```

6) Crear un Trigger en la tabla plantilla. Cuando actualicemos la tabla plantilla, debemos comprobar que el hospital que actualizamos existe, si intentamos actualizar el código de hospital, no podremos hacerlo si no existe relación con algún código de hospital. Realizar el mismo Trigger para las tablas relacionadas con Hospital.

```
Create
trigger ActualizarPlantilla on Plantilla
for update
as
declare @Hospital int
select @Hospital = i.hospital_cod
from hospital as h
inner join inserted as i
on h.hospital_cod = i.hospital_cod

if (@Hospital is null)
begin
    print 'No Existe el codigo de Hospital'

    update Plantilla set Hospital_cod = d.hospital_cod
    from Plantilla as h
    , inserted as i
    , deleted as d
    where h.hospital_cod = i.hospital_cod
end
Else
    print 'Existe el codigo en el hospital'
```

7) Modificar el Trigger del ejercicio 4, utilizando transacciones y control de errores, si la operación es correcta, mostrará un mensaje positivo, si la operación no es correcta mostrará el error y un mensaje que indique que no se ha llevado a cabo la operación.

```
Alter Trigger [DAR ALTA]
on Emp
For Insert
as
Declare @Error int
Begin tran
Insert into ControlTrigger (N_emp, Usuario, Fecha, Operación)
Select inserted.emp_no, user_name(), GetDate(), 'INSERCION'
From Inserted
Set @Error = @@ERROR
if @@Error <> 0
Begin
Rollback tran
print 'Existe un error en el Trigger'
print @@Error
end
else
begin
commit tran
print 'Empleado insertado correctamente'
end
```

8) Crear un Trigger que guarde los datos en la tabla ControlTrigger cuando se realice la baja de un empleado.

```
Create Trigger [DAR BAJA]
on EMP
For Delete
as
Insert Into ControlTrigger(N_EMP,USUARIO,FECHA,OPERACION)
Select Deleted.emp_no,USER_NAME(),GETDATE(),'BAJA'
from Deleted
```

9) Crear un Trigger que guarde los datos en la tabla ControlTrigger cuando se relice una modificación en un empleado. Guardar la hora de la actualización en un campo aparte en la tabla ControlTrigger. (Añadir un campo)

```
Create Trigger [ModificarEmp]
on Emp
For Update
as
DECLARE @HORA NVARCHAR(10)
SET @HORA = convert(char(2),datepart(hh,getdate())) + ':'
```

```
+ convert(char(2),datepart(mi,getdate()))
+ ':' + convert(char(2),datepart(ss,getdate()))
Insert Into ControlTrigger(N_EMP,USUARIO,FECHA,OPERACION,HORA)
Select Deleted.emp_no,User_Name(),GetDate(),'MODIFICACION',@HORA
from Deleted, Inserted
where deleted.emp_no = inserted.emp_no
```

10) Borrar todos los Triggers creados después de haber sido probados y volver a dejar la base de datos como estaba desde la copia de seguridad.

```
DROP TRIGGER .....
```

MÓDULO 3: Creación y Administración de bases de datos

Cada vez que generamos una base de datos la información de esta, se incluye dentro de la base de datos **Master** en la tabla **SysDataBases**. Si al crear la base de datos no establecemos una serie de valores para configurarla, toma como estos valores de la base de datos **Model**, la cuál sirve de modelo para crear una base de datos por defecto.
Una base de datos está compuesta de dos tipos de archivo:

1. Archivos de datos: Es donde están guardados los datos.
    a. .MDF
    b. .NDF: Contiene las copias de seguridad. También vistas, tablas y consultas que se usan mas frecuentemente. Este tipo de archivos se puede separar e instalar en diferente máquina que el resto del servidor de la base de datos, para poder instalarlo en una máquina mas potente.
2. Registro de transacciones: Es donde se cargan todas las consultas antes de ejecutarlas en los archivos de datos. Se ejecutan en este registro de transacciones y si el resultado es correcto, se ejecutan en los archivos de datos. También sirve como histórico ya que almacena todas las transacciones realizadas.
    a. LDF

❑ Crear una Base de Datos

Para ello usaremos el comando CREATE.

```
CREATE DATABASE BBDD
On
        Primary
        (Name = Nombre,
        Filename = 'Ruta\NombredeArchivo.MDF',
        Size = Tamaño por defecto,
        MaxSize = Tamaño máximo,
        Filegrowth = Porcentaje / tamaño de crecimiento)
        Log on
        (NAME = Nombre,
        Filename = 'Ruta\NombredeArchivo.LDF',
        Size = Tamaño por defecto
        MaxSize = Tamaño máximo
        FileGrowth = Porcentaje / tamaño de crecimiento)
```

- *Primary*: Esta parte es donde se define y configura el archivo MDF
- *Log On*: En esta parte es donde se define y configura el archivo LDF
- *Name*: Indica el nombre que tendrá el archivo MDF o LDF
- *Filename*: Indica la ruta donde estará almacenado el MDF o LDF y su nombre.
- *Size*: Indica el tamaño por defecto que tendrá el MDF o LDF
- *MaxSize*: Indica el tamaño máximo que puede alcanzar el MDF o LDF
- *FileGrowth*: Indica cuanto crecerá el archivo MDF o LDF cuando llegue a su tamaño máximo. Este tamaño lo podemos fijar mediante porcentaje, MB o KB. Si no establecemos esta opción, cuando el archivo alcance su tamaño máximo ya no podrá crecer mas.

```
CREATE DATABASE Ejemplo
On
        Primary
        (Name = EjemploData,
        Filename = 'C:\Ejemplo.MDF',
        Size = 5MB,
        MaxSize = 10MB,
        Filegrowth = 20%)
        Log on
        (NAME = EjemploLog,
        Filename = 'C:\EjemploLog.ldf',
        Size = 3MB,
        MaxSize = 5MB,
        FileGrowth = 1MB)
```

El proceso CREATE DATABASE está asignando 5.00 MB en el disco 'EjemploData'.
El proceso CREATE DATABASE está asignando 3.00 MB en el disco 'EjemploLog'.

- **SP_HELPDB**

  Muestra todas las bases de datos del servidor, con su propietario, tamaño hora de creación etc.

```
SP_HELPDB
```

	name	db_size	owner	dbid	created	status
3	CIUDADES	2.00 MB	A1S13\Administrador	11	Oct 4 2002	Status=ONL
4	Ejemplo	8.00 MB	sa	13	Oct 8 2002	Status=ONL
5	Ejemplo Microsoft Search	2.00 MB	A1S13\Administrador	12	Oct 5 2002	Status=ONL
6	Hospital	2.00 MB	A1S13\Administrador	9	Jul 9 2002	Status=ONL
7	master	51.13 MB	sa	1	Ago 6 2000	Status=ONL
8	model	1.25 MB	sa	3	Ago 6 2000	Status=ONL
9	msdb	14.00 MB	sa	4	Ago 6 2000	Status=ONL
10	Northwind	4.25 MB	sa	6	Ago 6 2000	Status=ONL
11	pubs	2.50 MB	sa	5	Ago 6 2000	Status=ONL
12	tempdb	8.50 MB	sa	2	Oct 8 2002	Status=ONL
13	VideoClub	2.75 MB	A1S13\Administrador	10	Sep 3 2002	Status=ONL

- **SP_SPACEUSED**

  Muestra el espacio usado de la base de datos que estamos usando actualmente.

```
USE EJEMPLO
Exec SP_SPACEUSED
```

	database_name	database_size	unallocated space
1	Ejemplo	8.00 MB	4.48 MB

	reserved	data	index_size	unused
1	536 KB	152 KB	280 KB	104 KB

Cuadrículas | Mensajes

- **Grupos de archivos secundarios**

Son los formados por archivos NDF, donde como hemos visto antes, se almacenan copias de seguridad, vistas etc.
Para crear un grupo de archivo secundario seguiremos los siguientes pasos:

1. Modificamos la base de datos para añadirle el grupo de archivos secundario, para ello usaremos Add FileGroup
   Alter Database Base de datos
   Add FileGroup Grupodearchivosecundario

   Alter Database Ejemplo
   Add FileGroup MuyConsultados

   Comandos completados con éxito.

2. Modificamos la base de datos para añadir un archivo NDF físico al grupo de archivos secundario, para ello usaremos Add File. Sintaxis:
   Alter Database Base de datos
   Add File
   (
   Name = ‘Nombre’,
   Filename = ‘Ruta\Archivo.Ndf’,
   Size = Tamaño en MB
)

   Alter Database Ejemplo
   Add File
   (
   Name = 'Consulta',
   Filename = 'C:\Consulta.Ndf',
   Size = 5 MB
)

   Ampliando la base de datos en 5.00 MB de disco 'Consulta'.

   - **SP_HelpFile**

     Muestra información sobre archivos NDF. Sintaxis:
     SP_HELPFILE @Filename = ‘Nombredearchivosinextensión’

     SP_HELPFILE @Filename = 'Consulta'

	name	filename	filegroup	size	maxsize	growth	usage
1	Consulta	C:\Consulta.Ndf	PRIMARY	5120 KB	Unlimited	10%	data only

- **Modificar y añadir archivos a grupos de archivos secundarios**

  Mediante la opción Modify File de Alter Database, podemos modificar un archivo secundario. Sintaxis:

  Alter Database Base de datos
  Modify File (name = ‘nombre’, propiedades y valores a modificar)

  Alter Database Ejemplo
  Modify File (name = 'ejemplolog', size = 15MB)

  Con Add File, podemos añadir un nuevo archivo NDF al archivo secundario de la base de datos elegido. Sintaxis:

  Alter Database Base de datos
  Add File
  (Name = 'Nombre',
  Filename = 'Ruta\Archivo.NDF',
  Size = Tamaño por defecto,
  Maxsize = Tamaño máximo)
  To Filegroup Grupo de archivos secundario

  Alter Database Ejemplo
  Add File
  (Name = 'Ejemplodata2',
  Filename = 'C:\Ejemplo2.NDF',
  Size = 10MB,
  Maxsize = 15MB)
  To Filegroup MuyConsultados
  Ampliando la base de datos en 10.00 MB de disco 'Ejemplodata2'.

Alter Database Clientes
Remove File 'RelacionAnual2'
Go

El archivo 'RelacionAnual2' se quitó.

Podemos modificar el tamaño de un archivo secundario mediante ShrinkFile.

Sintaxis:

DBCC ShrinkFile (Nombrearchivo, %tamañoareducir)

DBCC ShrinkFile (EjemploData, 10, Opciones)

Opciones:

- *EmptyFile*: Indica que el tamaño del archivo reducido, pase a otros archivos secundarios del mismo grupo.
- *TruncateOnly*: Indica que el tamaño reducido de la base de datos pasa al sistema operativo. Si no ponemos ninguna opción esta es la opción por defecto
- *NoTruncate*: Indica que el tamaño reducido de la base de datos no pasa al sistema operativo.

- **ShrinkDataBase**

  Modifica el tamaño de la base de datos. Sintaxis:

  Sintaxis:

  DBCC Shrinkdatabase (Basededatos, %tamañoareducir, Opciones)

  Opciones:

  - *EmptyFile*: Indica que el tamaño del archivo reducido, pase a otros archivos secundarios del mismo grupo.
  - *TruncateOnly*: Indica que el tamaño reducido de la base de datos pasa al sistema operativo. Si no ponemos ninguna opción esta es la opción por defecto
  - *NoTruncate*: Indica que el tamaño reducido de la base de datos no pasa al sistema operativo.

  Ejemplo:

  DBCC Shrinkdatabase (Ejemplo, 25, NoTruncate)

Creación de Bases de Datos

1º) Crear una base de datos de Clientes con un tamaño de 40 MB y con crecimiento de un 10%.

```
create database Clientes
on
primary
(Name=Clientes_Data, Filename='C:\ Base\ClientesData.MDF',size=25MB,
Maxsize=100MB, Filegrowth=10%)
log on
(Name=Clientes_Log, Filename='C:\ Base\ClientesLog.LDF',size=15MB,
Maxsize=40MB, Filegrowth=10%)
```

El proceso CREATE DATABASE está asignando 25.00 MB en el disco 'Clientes_Data'.
El proceso CREATE DATABASE está asignando 15.00 MB en el disco 'Clientes_Log'.

2º) Mostrar si la base de datos está junto a las demás y el espacio usado de mi nueva base de datos.

```
sp_helpdb
sp_spaceused
```

3º) Crear un grupo de archivos en mi base de datos clientes que se llame Bajas. Añadir dos archivos a ese grupo de archivos llamados RelacionAnual y que ocupen 5MB.

```
alter database Clientes
add filegroup Bajas

alter database Clientes
add file
(name='RelacionAnual1', filename='c:\RelacionAnual1.NDF', size=5MB)
to filegroup Bajas

alter database Clientes
add file
(name='RelacionAnual2', filename='c:\RelacionAnual2.NDF', size=5MB)
to filegroup Bajas
```

4º) Mostrar las características de los archivos que acabamos de incluir y de su grupo.

```
Exec sp_Helpfile @filename='RelacionAnual1'
Exec sp_Helpfile @filename='RelacionAnual2'
Exec sp_helpfilegroup 'Bajas'
```

5º) Eliminar uno de los archivos del grupo Bajas y modificar el otro para darle un tamaño de 10 MB.

```
alter Database Clientes
remove file 'RelacionAnual2'

alter Database Clientes
modify file (Name='RelacionAnual1', size=10Mb)
```

6º) Reducir el tamaño de la base de datos en un 2%.

```
Dbcc shrinkdatabase(Clientes, 2)
```

7º) Volver a aumentar el tamaño de la base de datos en 5 MB más.

```
alter Database Clientes
modify file (Name='Clientes_Data', size=5)
```

8º) Poner el tamaño máximo de la base de datos a 60 MB.

```
alter Database Clientes
modify file (Name='Clientes_Data', maxsize=20)
modify file (Name='Clientes_Log', maxsize=20)
modify file (Name='RelacionAnual1', maxsize=20)
```

9º) Eliminar el grupo de Bajas de la base de datos junto con sus archivos. Eliminar también la base de datos.

```
alter Database Clientes
remove file relacionanual1

alter Database Clientes
remove filegroup bajas

drop database Clientes
```

MODULO 8: Consultas de índices de texto

- **Servicio Microsoft Search**

Permite realizar búsquedas en las tablas de datos. Es muy potente y rápido para buscar en una base de datos documental, ya que no solo realiza búsquedas exactas por palabras y caracteres, sino también por aproximación e incluso sinónimos (solo para palabras en inglés). Esto se realiza mediante catálogos de texto, estos catálogos, contienen a su vez índices de texto, los cuales tienen las palabras características de cada tabla.
Para que una tabla pueda estar en un índice de texto, a de tener una sola clave principal, no admitir nulos. **Además solo podemos indexar en campos de tipo texto ya que realiza búsquedas por caracteres y palabras.**

Crear la siguiente tabla en SQL para realizar el ejercicio:

```
create table Libros
(Identidad int identity(1,1) constraint PK_Libros primary key
,Titulo nvarchar(100) null
,Descriptores nvarchar(100) null
,Comentario nvarchar(100) null)


insert into Libros (Titulo,Descriptores,Comentario)
values ('LA COLMENA','FICCION ESPAÑA DRAMA','MUY BUENO')
insert into Libros (Titulo,Descriptores,Comentario)
values ('HISTORIA DE ROMA','ENSAYO ITALIA HISTORIA','MUY
INTERESANTE')
insert into Libros (Titulo,Descriptores,Comentario)
values ('METAFISICA','ENSAYO GRECIA FILOSOFIA
CLASICA','INTERESANTE')
insert into Libros (Titulo,Descriptores,Comentario)
values ('EL NOMBRE DE LA ROSA','FICCION EDAD MEDIA INTRIGA','MUY
BUENO')
insert into Libros (Titulo,Descriptores,Comentario)
values ('LA REPUBLICA','ENSAYO GRECIA FILOSOFIA
CLASICA','INTERESANTE')
insert into Libros (Titulo,Descriptores,Comentario)
values ('VISUAL BASIC NET PARA NIÑOS','REFERENCIA INFORMATICA
PROGRAMACION','REGULAR')
insert into Libros (Titulo,Descriptores,Comentario)
values ('EL PERFUME','FICCION INTRIGA','MUY BUENO')
insert into Libros (Titulo,Descriptores,Comentario)
values ('INNER JOIN POR LAS MAÑANAS','REFERENCIA INFORMATICA
PROGRAMACION','REGULAR')
```

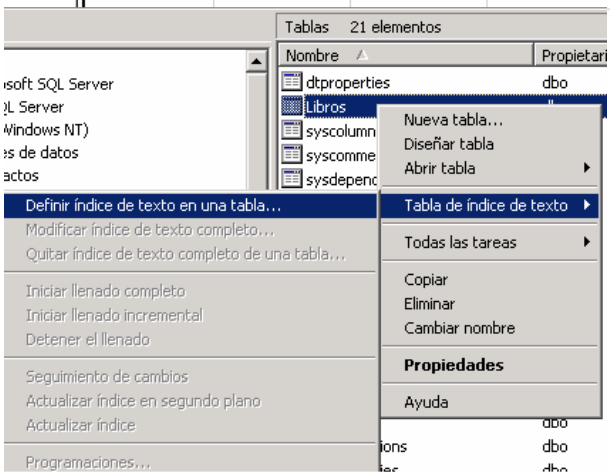
Como crear un catálogo de texto

1. Creamos la tabla con sus campos, un campo principal que no admita nulos, y los campos que van a ser indexados para los catálogos tipo carácter.

	Nombre de columna	Tipo de datos	Longitud	Permitir valores nulos
🔑	ID	int	4	
	TITULO	nvarchar	100	✓
	DESCRIPTORES	nvarchar	100	✓
	COMENTARIO	nvarchar	100	✓

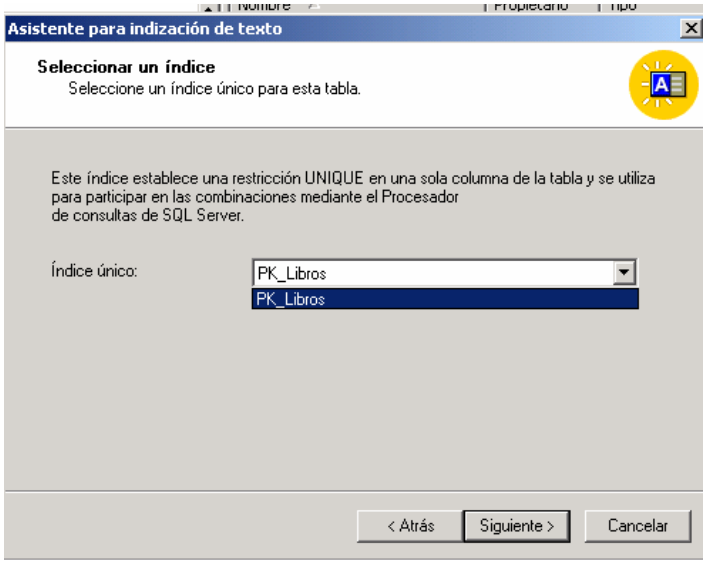
2. Introducimos los datos que va a contener la tabla.

ID	TITULO	DESCRIPTORES	COMENTARIO
1	LA COLMENA	FICCION ESPAÑA D	MUY BUENO
2	HISTORIA DE ROM	ENSAYO ITALIA HI	MUY INTERESANTE
3	METAFISICA	ENSAYO GRECIA F	INTERESANTE
4	EL NOMBRE DE LA I	FICCION EDAD MEI	MUY BUENO
5	LA REPUBLICA	ENSAYO GRECIA F	INTERESANTE
6	VISUAL BASIC NET	REFERENCIA INFO	REGULAR
7	EL PERFUME	FICCION INTRIGA	MUY BUENO
8	INNER JOIN POR L	REFERENCIA INFO	REGULAR



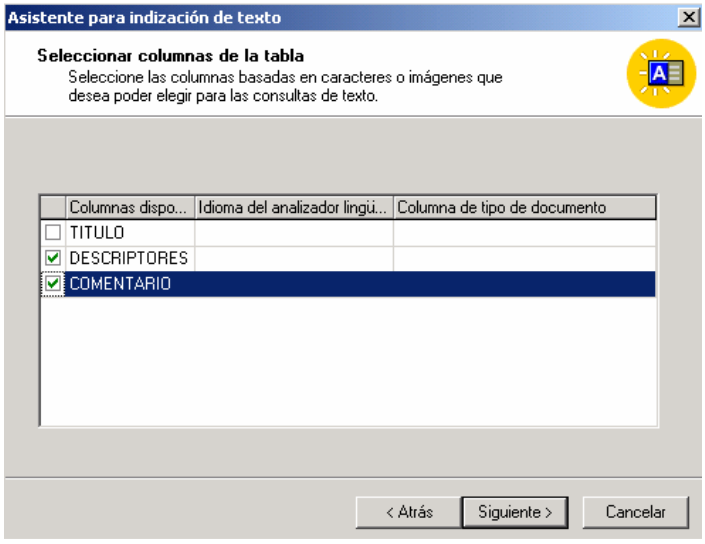
3. En la tabla que hemos creado, pulsamos botón derecho y seleccionamos:
   Tabla de índice de texto -> **Definir índice de texto en una lista**

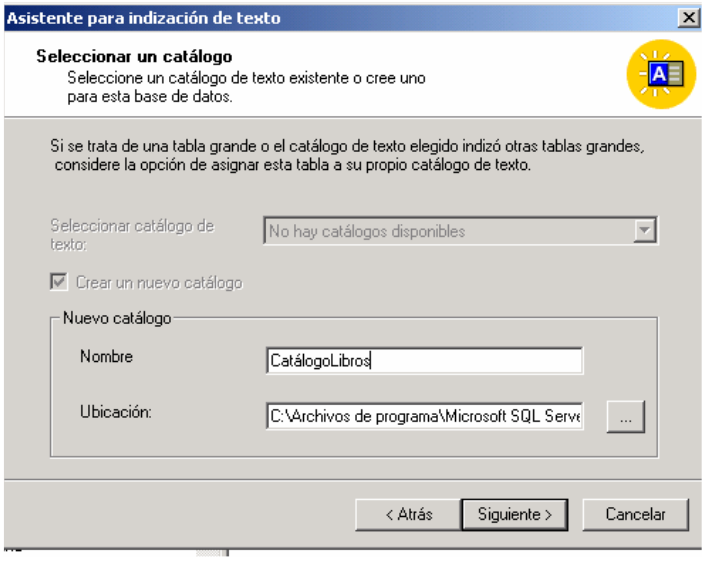
4. Se ejecutará el **Asistente para indización de texto**. Pulsamos siguiente.



5. Seleccionamos el índice de la tabla, necesario para realizar las búsquedas.
   Pulsamos Siguiente.

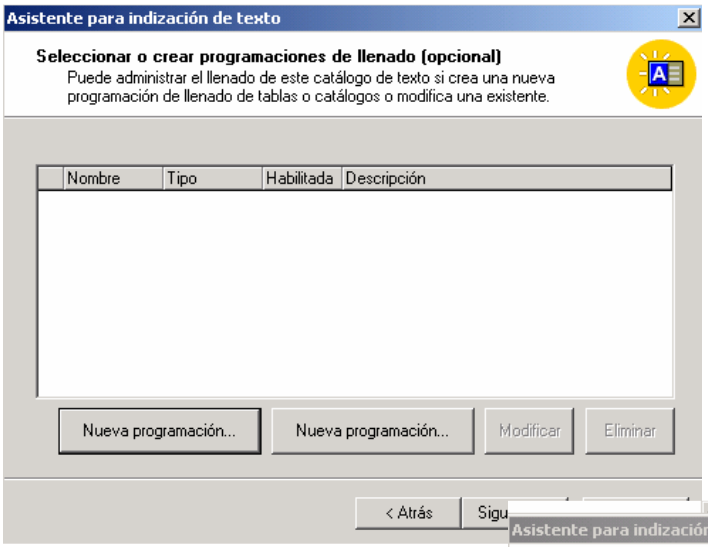
6. Seleccionamos las columnas **de texto** sobre las que se realizarán los catálogos de búsqueda.
   Pulsamos Siguiente.





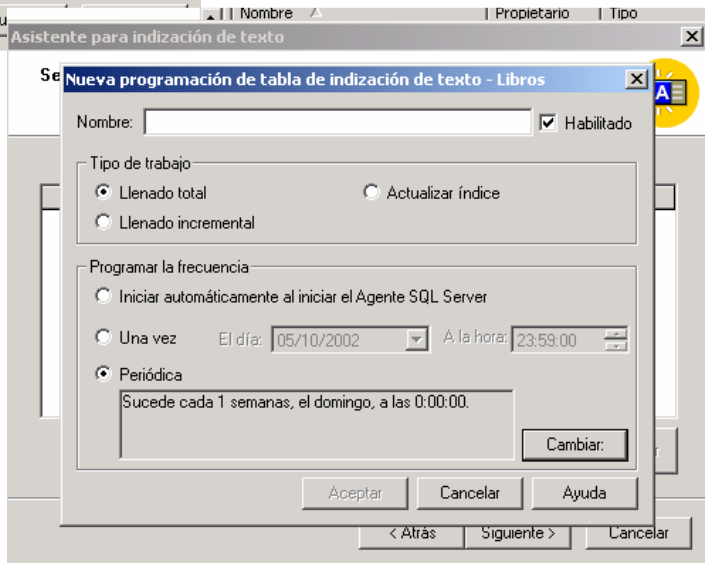
7. En esta pantalla deberemos seleccionar el catálogo en el que se van a almacenar los índices de texto. En este caso, como no existe ningún catálogo crearemos un nuevo. Eligiendo su nombre y dejando la ubicación por defecto.

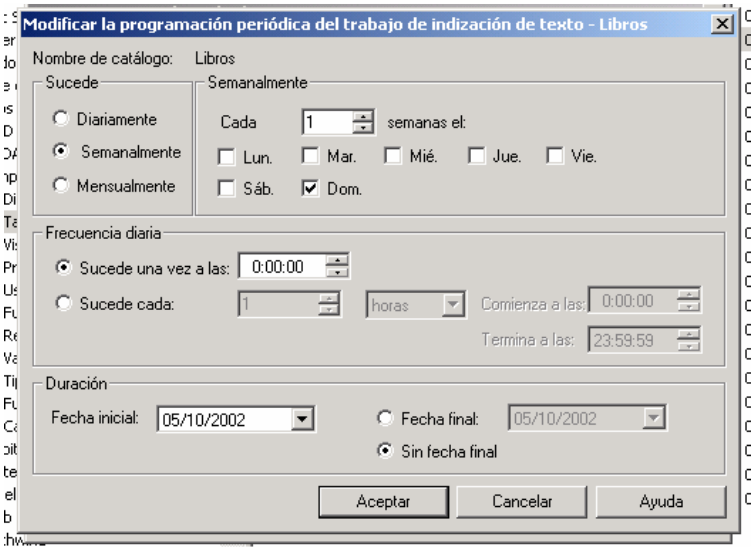
Posteriormente, al indizar otras tablas, podremos elegir el mismo catálogo creado o si vemos que los datos que van a almacenar son muy extensos, crear uno nuevo explicito para esa tabla.



8. Esta pantalla es opcional. Se usa para programar el llenado de los catálogos de texto, por si la información que contienen las tablas se actualiza.
Si pulsamos Nueva programación...

9. ...nos saldrá una pantalla en la que podremos configurar la forma del llenado y la frecuencia.
Si seleccionamos **frecuencia periódica** y posteriormente pulsamos el botón **Cambiar**...

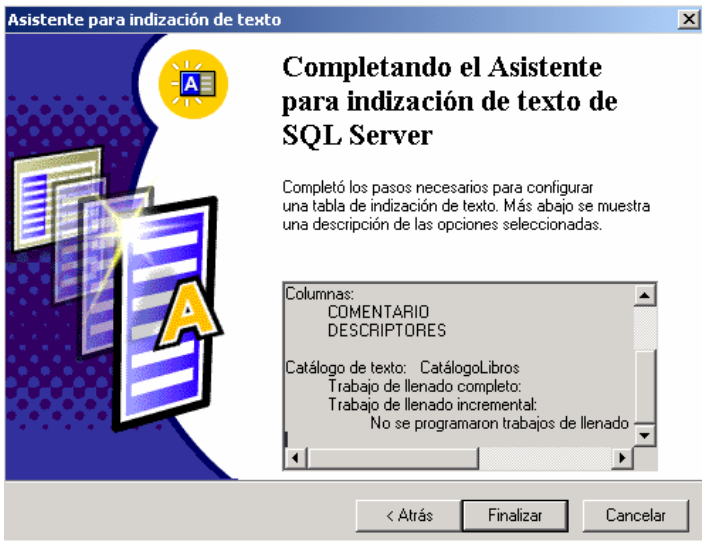


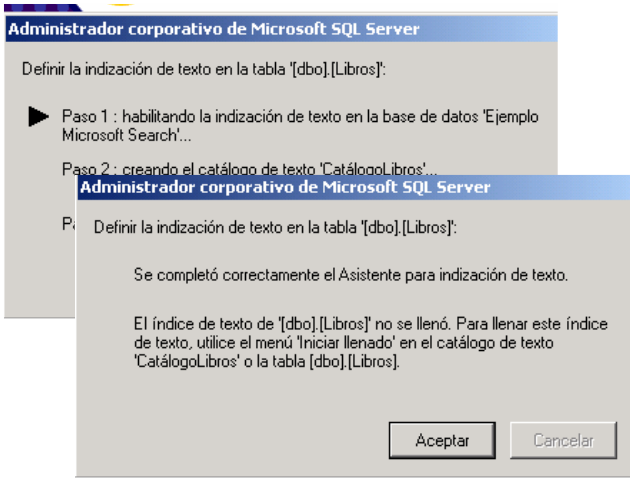


10. ...aparecerá una pantalla donde configuraremos la frecuencia (periocidad, día de la semana, hora, intervalo de fechas que dura la programación etc.)

Pulsamos Cancelar hasta volver a la pantalla **Seleccionar o Crear programación de llenado**(paso nº 8). Una vez en esta pantalla, pulsamos Siguiente.

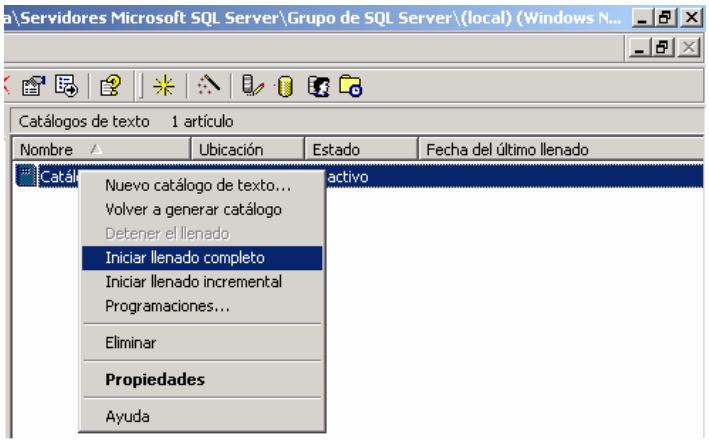
11. Ya se ha completado los pasos necesarios para indizar la tabla "Libros". Pulsamos **Finalizar**.

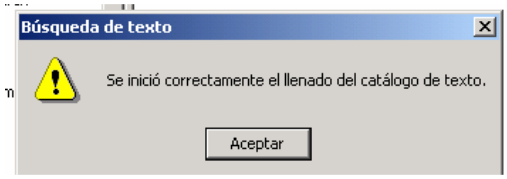




12. Aparecerá una pantalla en la que iremos viendo como se van realizando los pasos para indizar el tabla e introducirla en el catálogo. Una vez finalizada la indización pulsamos **Aceptar.**

13. Posteriormente, dentro la base de datos con la que estamos trabajando, hacemos click sobre **Catálogos de Texto**. Veremos que está el catálogo que acabamos de crear. Debido a que no lo hemos llenado con el asistente, lo llenaremos mediante la opción **Iniciar llenado completo**.





14. Una vez finalizado el llenado, veremos el mensaje de la imagen.

Después realizar todos estos pasos, ya se habrán creado los catálogos de texto

Procedimientos almacenados del sistema para catalogos de texto.

Una vez creados y llenado los catálogos, debemos activarlos con procedimientos almacenados del sistema. Todos los procedimientos almacenados del sistema empiezan por SP
SP_FullText_Database: Activa / Desactiva todos los índices de todos los catálogos de texto para poder realizar las búsquedas con Microsoft Search. Sintaxis

SP_FullText_Database 'Enable'
SP_FullText_Database 'Disable'

SP_help_FULLTEXT_COLUMNS Indica las tablas y las columnas que están indexadas mediante catálogos (ejecutado sin parámetros). Sintaxis:
sp_help_fulltext_columns [[**@table_name** =] '*table_name*']]
[**,** [**@column_name** =] '*column_name*']

Argumentos

[**@table_name** =] '*table_name*'

Es el nombre de tabla de una o dos partes para el que se solicita información de índice de texto. El argumento *table_name* es de tipo **nvarchar(517)** y tiene el valor predeterminado NULL. Si se omite *table_name*, se obtiene la información de columna de índice de texto de todas las tablas con índice de texto.

[**@column_name** =] '*column_name*'

Es el nombre de la columna cuyos metadatos de indización de texto se solicitan. El argumento *column_name* es de tipo **sysname** y su valor predeterminado es NULL. Si se omite el argumento *column_name* o es NULL, se obtiene la información de la columna de texto para cada columna con índice de texto del argumento *table_name*. Si también se omite *table_name* o es NULL, se obtiene la información de la columna de índice texto para cada columna con índice de texto de todas las tablas de la base de datos.

Funciones del sistema

Podemos utilizar funciones de transact-sql para obtener valores de las propiedades de texto.
Select **DatabaseProperty** ('Base de datos', **IsFullTextEnabled**)
Muestra si la base de datos indicada está preparada para la búsqueda de texto. Devuelve NULL si no está preparada y 1 si está preparada.

Ejemplo:

```
select databaseproperty('Ejemplo Microsoft Search','Isfulltextenabled')
```

Realizar búsquedas en un catalogo de texto

Instrucción Contains

Realiza búsquedas en columnas de tipo carácter, pueden ser coincidencias exactas, o aproximadas. Es capaz de buscar:

- Una palabra o una frase.
- El prefijo de una palabra o una frase.
- Una palabra cerca de otra palabra.
- Una palabra que sea una inflexión de otra, por ejemplo, las palabras controles, controladores, controlando y controlado son inflexiones de control (solo palabras en inglés).
- Una palabra que tenga un peso especificado mayor que el de otra.

Búsquedas exactas o aproximadas: 'PALABRA O FRASE'

Búsquedas exactas: '"Palabra o frase"'

Combinación de elementos: Podemos usar AND y OR. 'Palabra o frase OR Palabra o frase'

```
Select titulo, descriptores
from Libros
where contains (descriptores, 'FICCION')
```

	titulo	descriptores
1	EL PERFUME	FICCION INTRIGA
2	EL NOMBRE DE LA ROSA	FICCION EDAD MEDIA INTRIGA
3	LA COLMENA	FICCION ESPAÑA DRAMA

```
Select titulo, descriptores
from Libros
where contains (descriptores, '"EDAD MEDIA"')
```

	titulo	descriptores
1	EL NOMBRE DE LA ROSA	FICCION EDAD MEDIA INTRIGA

```
Select titulo, descriptores
from Libros
where contains (descriptores, 'HISTORIA OR FILOSOFIA')
```

	titulo	descriptores
1	LA REPUBLICA	ENSAYO GRECIA FILOSOFIA CLASICA
2	METAFISICA	ENSAYO GRECIA FILOSOFIA CLASICA
3	HISTORIA DE ROMA	ENSAYO ITALIA HISTORIA

```
Select titulo, descriptores
from Libros
where contains (descriptores, '"IN*"')
```

	titulo	descriptores
1	EL PERFUME	FICCION INTRIGA
2	VISUAL BASIC NET PAR...	REFERENCIA INFORMATI...
3	EL NOMBRE DE LA ROSA	FICCION EDAD MEDIA I...
4	INNER JOIN POR LAS M...	REFERENCIA INFORMATI...

```
Select titulo, descriptores
from Libros
where contains (descriptores, 'ENSAYO NEAR CLASICA')
```

	titulo	descriptores
1	LA REPUBLICA	ENSAYO GRECIA FILOSOFIA CLASICA
2	METAFISICA	ENSAYO GRECIA FILOSOFIA CLASICA

```
Select titulo, descriptores
from Libros
where contains (descriptores, 'INFORMATICA NEAR PROGRAMACION')
```

	titulo	descriptores
1	VISUAL BASIC NET PARA NIÑOS	REFERENCIA INFORMATICA PROGRAMACION
2	INNER JOIN POR LAS MAÑANAS	REFERENCIA INFORMATICA PROGRAMACION

FreeText: Es como un buscador, busca coincidencias eliminando pronombres.

```
select titulo
,descriptores
from libros where Freetext(Descriptores,'ENSAYO DE LA INFORMATICA')
```

TEMA 2 CREACION DE TIPOS DE DATOS Y TABLAS

1. **Creación de tipos de datos definidos por el usuario:** Se basan en tipos de datos definidos por el SQL. Para crear un tipo de dato se usa un procedimiento almacenado del sistema.

Administrador Corporativo → Eliges una Tabla → Procedimientos almacenados que empiecen por sp

Para crear un nuevo tipo de dato se usa sp_addtype

sp_addtype nombre, tipo de dato, [‘NULL’ | ‘NOT NULL’], [propietario]

EXEC sp_addtype Ciudad, ‘nvarchar (15)’, ‘NULL’

Con esto borras el dato

EXEC sp_droptype Ciudad

2. **Creación y eliminación de una tabla**

El valor por defecto es NULL

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

Para eliminarla

DROP TABLE NombreTabla

3. **Agregar y quitar 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

Para quitar la columna

ALTER TABLE NombreTabla DROP COLUMN NombreColumna

ALTER TABLE emp DROP COLUMN Edad

4. Generación de valores de columnas

- Uso de propiedad IDENTITY

  Es un tipo de dato que genera valores y los incrementa. Un contador vamos. Solo una columna con Identity, el tipo de dato debe ser entero int, numerico numeric o decimal, estos dos últimos con escala cero. Debe ser obligatorio que sea NOT NULL. Inicial e incremento son opcionales y por defecto comenzarian en 1,1.

  CREATE TABLE Tabla (Columna TipoDato IDENTITY([inicial, incremento]) NOT NULL)

```
create table NINONINO
(
identidad int identity(1,1) not null
,apellido nvarchar(10) null
)
```

- Uso función NEWID y tipo de datos uniqueidentifier

Pueden cambiarse sin volver a crear la tabla
Requieren comprobación de errores
Comprueban datos existentes

Default

- Se aplica para INSERT
- Solo una por columna
- No se pueden utilizar con IDENTITY
- Permite especificar valores proporcionados por el sistema

```
alter table NINONINO
add constraint DF_Apellido
default 'SERRANIN'
for apellido
```

sintaxis parcial

[constraint nombre_restricción] default expresión

CHECK

- Para sentencias INSERT y UPDATE
- Pueden hacer referencias a columnas en la misma tabla
- No pueden contener subconsultas

Sintaxis parcial
[Constraint nombrerestriccion)
chech (Expresion)

Primary key

- Solo una por tabla
- Valores exclusivos
- No se permiten valores nulos
- Crea un índice exclusivo en las columnas especificadas
  (Predet. --> CLUSTERED)

Sintaxis parcial
[Constraint NombreRestic]
primary key [Clustered | NonClustered] columnas,…)

```
alter table emp
add
constraint Pk_Emp
primary key nonclustered (emp_no)
```

Unique

- Permite un valor nulo
- Permite varias en una tabla
- Formadas por una o mas columnas
- Exigida con un índice único

Sintaxis parcial
[Constraint nombreRestriccion]
Unique [Clustered | non Clustered] (Columnas,...)

```
alter table emp
add
constraint U_Apellido
Unique Nonclustered (Apellido)
```

Foreign key

- Deben hacer referencia a PRIMARY KEY o UNIQUE
- Proporcionan integridad referencial de una o varias columnas
- No crean índices automáticamente
- Permisos SELECT o REFERENCES

Sintaxis parcial

[Constraint nombreRest]
Foreign key (Columna,...)
References tablaRef (Columna,…)

```
alter table Emp
add
Constraint Fk_Emp_Dept
Foreign Key (dept_no)
References Dept (dept_no)
```

Realizo una referencia a la tabla Departamento y el campo Dept_no para crear una restricción que solamente me permita insertar empleados que esten en un departamento de la tabla Dept.

Característica orientada a las claves externas

Integridad referencial en cascada

Sintaxis parcial

[Constraint NombreRestriccion)
[Foreign Key] (Columna)
References TablaRef (Columnas,…)
[On Delete {Cascade | No Action}]
[On Update {Cascade | No Action}]

- NO Action es la predeterminada

	Products		Orders
PK	ProductId	PK	OrderId
		FK	ProductId

Con cascade si borras , borrara lo que este asociado a ella en la otra tabla , pero con No Action no te dejara borrarlo ya que esta asociado a otra tabla la cual tiene la clave principal. . Con cascade puedes borrar ProductId de products , lo cual borrara productId de orders , con No Action no lo podras borrar

```
alter table Emp
add
Constraint Fk_Emp_Dept
Foreign Key (dept_no)
References Dept (dept_no)
On delete Cascade
On update Cascade
```

Con estas dos instrucciones al final de la restricción le estoy diciendo que borre todos los empleados de un departamento si dicho departamento es eliminado de la tabla Dept. Lo mismo sucederá al modificar.

Deshabilitacion de restricciones

- Deshabilitación para la comprobación de los datos existentes
  - Aplicable a CHECK y FOREIGN KEY
  - Utilizar la opción WITH NOCHECK

Sintaxis parcial

Alter Table
[WITH CHECK | WITH NOCHECK]
Add
Constrain restriccion
{ Foreign key (Columna,…)
References TablaRef (Columna,…)
{ CHECK (Condicion)

Restricción de una tabla consigo misma

```
use northwind
alter table employees
with NOCHECK
add
Constraint Fk_employees_employees
Foreign Key (Reportstu)
References employees(EmployeeId)

alter table emp2
with NOCHECK
add
Constraint FK_Emp_Dir
Foreign Key (Dir)
References emp2(Emp_no)
```

- Deshabilitación al cargar nuevos datos
  - Los datos verifican la restricción
  - Datos que no cumplen la restricción pero que se quieren guardar en la tabla

Sintaxis parcial

Alter Table tabla
{CHECK | NOCHECK } CONSTRAINT
{ALL | Restricción 1, … }→ Para hacer la restriccion a todo , o a solo lo que esta separado por comas “,”

Deshabilita la restricción hecha antes

```
use northwind
alter table employees
NOCHECK CONSTRAINT FK_Employees_Employees
```

Con esto introduces los datos y vuelves a habilitar la restricción

- Solamente en restricciones de tipo Check y Foreign Key

Valores predeterminados y las reglas como objetos independientes

Se van a crear valores predeterminados y reglas

- Se definen una vez
- Pueden vincularse a una o mas columnas o a tipos de datos definidos por el usuario
- Toda regla asociada a la columna y a los tipos de datos valida el valor de un valor predeterminado
- Cualquier restricción CHECK sobre la columna debe validar el valor de un valor predeterminado
- No se puede crear una restricción DEFAULT en una columna que este definida con un tipo de datos definido por el usuario si dicha columna o tipo tuvieran un valor predeterminado

Sintaxis

CREATE DEFAULT Nombre AS Expresión

Procedimientos de sistema para valores predeterminados

- **sp_helpconstraint** muestra todas las restricciones que tiene una tabla
- **sp_bindefault:** Con este procedimiento lo que hago es vincular el valor por defecto a la tabla.
- **sp_unbindefault:** Con este otro lo que hago es desvincular el valor por defecto de la tabla

```
use northwind
GO
CREATE DEFAULT Telefono_default AS '(00)000-0000'
GO

EXEC sp_bindefault Telefono_default,'Customers.Phone'
```

Al desvincularlo de la tabla se invierten los valores

```
EXEC sp_unbindefault 'Customers.Phone', Telefono_default

use northwind
EXEC sp_helpconstraint products
```

Para eliminar varios objetos de la base de datos con una sola sentencia

DROP DEFAULT Nombre,...

REGLAS

- Puede contener cualquier expresión valida para una clausula WHERE
- Una columna o tipo de dato definido por el usuario solo puede tener asociado una regla

Sintaxis

CREATE RULE NombreRegla
AS Condición

Procedimientos almacenados de sistema para las reglas:

- **sp_bindrule:** Vincula una regla a la tabla
- **sp_unbindrule:** Desvincula una regla de la tabla

```
use Hospital
create rule Funcion_Rule as
@Funcion in('INTERINO','ENFERMERO','ENFERMERA')
GO
EXEC sp_bindrule Funcion_rule,'Plantilla.Funcion'
```

Para desvincular la regla de la tabla se invierten los valores

```
EXEC sp_unbindrule 'Plantilla.Funcion', Funcion_rule
```

Si fuese un intervalo de valores se podría hacer asi

```
use northwind
GO
CREATE RULE regioncode_rule
AS
@regioncode >=1000 and @regioncode <=100
GO
EXEC sp_bindrule regioncode_rule,'Customers.Region'
```

O también

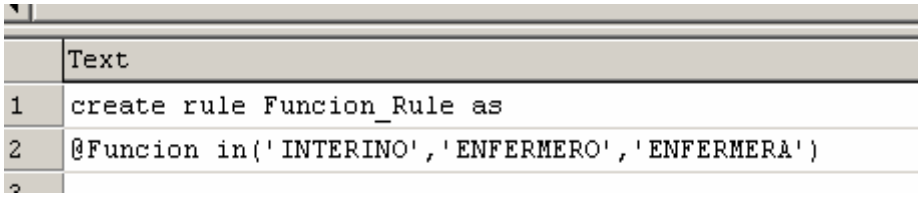
```
use northwind
GO
CREATE RULE regioncode_rule
AS
@regioncode like ‘[0-9][0-9 ][ 0-9]’
GO
EXEC sp_bindrule regioncode_rule,'Customers.Region'
```

Para borrar la regla se pone

```
DROP RULE NombreRegla
DROP RULE Funcion_rule
```

El primero de los procedimientos muestra el texto que se ha utilizado para crear la regla, el segundo procedimiento renombra el objeto, se puede utilizar con cualquier tipo de objeto de la base de datos.

```
Exec sp_helptext Funcion_Rule
```



	Text
1	create rule Funcion_Rule as
2	@Funcion in('INTERINO','ENFERMERO','ENFERMERA')

```
Exec sp_rename Oficio_rule, Funcion_Regla
```

Advertencia: al cambiar cualquier parte del nombre de un objeto pueden dejar de ser válidas secuencias de comandos y procedimientos almacenados.
El object ha cambiado su nombre por 'Funcion_Regla'.

Tabla para evaluar la integridad de los datos

	Funcionalidad	**Costos**	**AntesDespues Transacción**
Restricciones	Media	Baja	Antes
Valores predeterminados y reglas	Baja	Baja	Antes
Desencadenadores	Alta	Medio-Alto	Después (Excepto INSTEAD OF)
Tipos de Datos ,NULL | NOT NULL	Baja	Baja	Antes

Estructuras de las Tablas

1. Crear una nueva base de datos con un tamaño de 2 MB llamada Escuela.

```
create database Escuela
on primary
(name = Escuela_Data
, filename = 'C:\Escuela_Data.mdf'
,size = 1MB
,maxsize = 2MB
, filegrowth = 2%)
log on
(name = Escuela_Log
, filename = 'C:\Escuela_Log.ldf'
,size = 1MB
,maxsize = 2MB
,filegrowth = 2%)
go
```

2. Crear tipos de datos que contengas los datos más comunes para nuestra base de datos. (Mirar las bases de datos y cuales son sus campos más comunes).

Comprobar si se han introducido correctamente estos datos, mostrando solamente los Tipos de dato que hayamos introducido.

```
use Escuela
GO
exec sp_addtype TNoNulo,'nvarchar(15)','Not null'
exec sp_addtype TNulo,'nvarchar(15)','Null'

select * from systypes where name in('TNoNulo','TNulo')
```

name	xtype	status	xusertype	length	xprec	xscale	tdefault	domain	uid	reserved	collationid	usert
TNoNulo	231	3	257	30	0	0	0	0	1	0	53288	257
TNulo	231	2	258	30	0	0	0	0	1	0	53288	258

Tipos de Datos

TNoNulo → Va a utilizarse para los campos de Texto no nulos
TNulo → Se utilizará en los campos de Texto restantes

- Crear la tabla COLEGIOS con los siguientes campos:

Campos	Tipo de dato	Restricción
Cod_colegio	Numérico	Clave Principal e Identidad
Nombre	Texto	No permite nulos.
Localidad	Texto	
Provincia	Texto	
Año_Construcción	Fecha	
Coste_Construcción	Moneda	
Cod_Region	Numérico	
Unico	Unico	Clave Unica

```
create table colegios
(cod_colegio int identity(1,1) not null
constraint PK_Clave primary key
,nombre TNoNulo
,localidad TNulo
,provincia TNulo
,año_construccion smalldatetime null
,coste_construccion money null
,cod_region int null
,Unico uniqueidentifier default Newid()
)
```

- Crear la tabla PROFESORES con los siguientes campos:

Campos	Tipo de dato	Restricción
Cod_Profe	Texto	Clave Principal y no permite nulos.
Nombre	Texto	No permite nulos.
Apellido1	Texto	
Apellido2	Texto	
Edad	Numérico	
Localidad	Texto	
Provincia	Texto	
Salario	Money	
Cod_Colegio	Numérico	Clave ajena: COLEGIOS

```
create table profesores
(cod_profe TNoNulo
constraint PK_Profesores primary key
,nombre TNoNulo
,apellido1 TNulo
,apellido2 TNulo
,edad int null
,localidad TNulo
,provincia TNulo
,salario money
,cod_colegio int null
constraint FK_Profesores foreign key(cod_colegio) references colegios(cod_colegio))
```

- Crear la tabla REGIONES con los siguientes campos:

Campos	Tipo de dato	Restricción
Cod_Region	Numérico	Clave Principal e Identidad
Regiones	Texto	No permite nulos.

```
create table regiones
(cod_region int identity(1,1) not null
constraint PK_Regiones primary key
,regiones TNoNulo)
```

- Crear la tabla ALUMNOS con los siguientes campos:

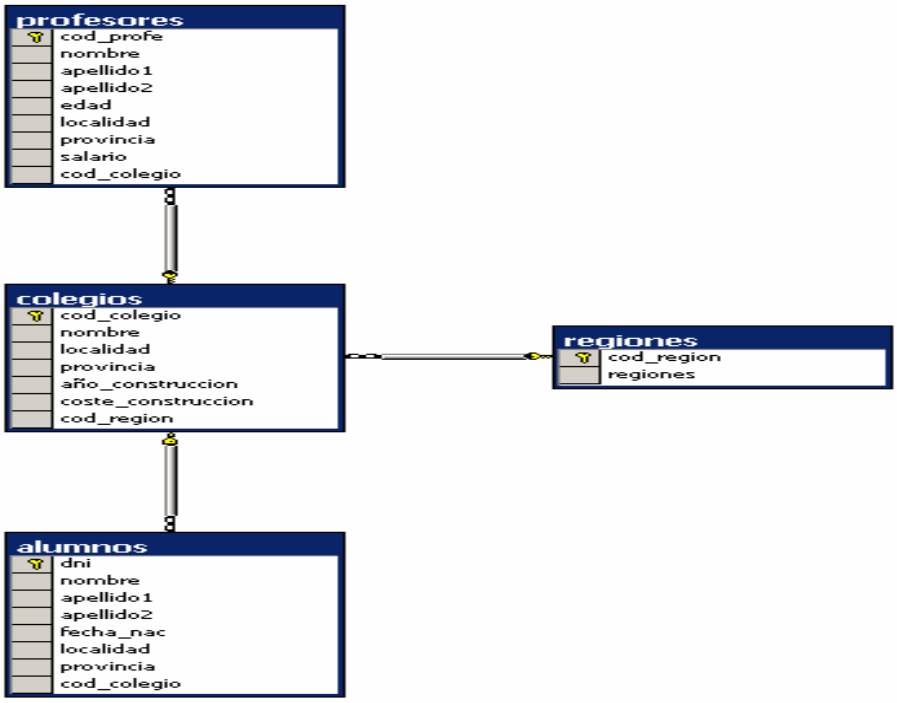
Campos	Tipo de dato	Restricción
DNI	Texto	Clave Principal y no permite nulos.
Nombre	Texto	No permite nulos.
Apellido1	Texto	
Apellido2	Texto	
Fecha_nac	Fecha/hora	
Localidad	Texto	
Provincia	Texto	
Cod_Colegio	Numérico	Clave ajena: COLEGIOS(Cod_colegio)

```
create table alumnos
(
dni TNoNulo
constraint PK_Alumnos primary key
,nombre TNoNulo
,apellido1 TNulo
,apellido2 TNulo
,fecha_nac smalldatetime
,localidad TNulo
,provincia TNulo
,cod_colegio int
constraint FK_Alumnos foreign key(cod_colegio)
references colegios(cod_colegio)
)
```

- Crear una nueva relación entre el campo Cod_Region de la tabla REGIONES y Cod_Region de la tabla colegios.

```
alter table colegios
add constraint FK_colegios
foreign key(cod_region)
references regiones(cod_region)
```

- Crear un gráfico que muestre las relaciones que se han obtenido en la base de datos hasta ahora.



- Añadir el campo Sexo, Fecha de nacimiento y Estado Civil a la tabla Profesores.

```
Alter Table Profesores
add Sexo nvarchar(6) null
,Fecha_Nac smalldatetime null
,[Estado Civil] nvarchar(12) null
```

- Eliminar el campo Edad de la tabla Profesores.

```
alter table Profesores
drop column Edad
```

- Añadir el campo Sexo, Dirección y Estado Civil a la tabla Alumnos.

```
alter table Profesores
drop column Edad
```

- Borrar la relación existente entre la tabla profesores y Colegios.

```
sp_helpconstraint profesores
sp_helpconstraint colegios
alter table profesores
drop constraint fk_profesores
```

- Crear de nuevo la relación borrada en el ejercicio anterior que tenga relación en cascada y actualización.

```
alter table profesores
add constraint FK_Profesores foreign key(cod_colegio)
references colegios(cod_colegio) on update cascade
on delete cascade
```

- Agregar un identificador único en la tabla Regiones.

```
alter table regiones
add Unico uniqueidentifier default newid()
```

- Queremos rellenar los datos de las tablas creadas, para ello vamos a introducir a tres alumnos en los colegios. Los datos son los siguientes:

Alumnos:
Marta Serrano Lopez Provincia: Madrid Localidad: Madrid
Javier Morgado Palomo Provincia: Alicante Localidad: Arenales del sol
Miguel Torres Tormo Provincia: Barcelona Localidad: Llobregat
Rellenar las tablas convenientemente con datos relacionales.
Ejemplo: Region: Comunidad Valenciana
3 datos como mínimo en cada tabla.

TABLA REGIONES

```
Insert into Regiones (Regiones)
values('MADRID')

Insert into Regiones (Regiones)
values('COMUNIDAD VALENCIANA')

Insert into Regiones (Regiones)
values('CATALUÑA')
```

Nos dará error por el tipo de dato introducido en el campo regiones, debemos cambiar el tipo de dato para poder utilizar la tabla

```
ALTER TABLE REGIONES
ALTER COLUMN REGIONES NVARCHAR(50) NULL
```

TABLA COLEGIOS

```
insert into Colegios
(nombre,localidad
,provincia,año_construccion
,coste_construccion,cod_region)
values
('PADRE POVEDA','MADRID','MADRID','01/01/65',1129876,1)
```

```
insert into Colegios
(nombre,localidad
,provincia,año_construccion
,coste_construccion,cod_region)
values
('CARMELITAS','ALICANTE','ELCHE','01/01/49',298763,2)

insert into Colegios
(nombre,localidad
,provincia,año_construccion
,coste_construccion,cod_region)
values
('SAN PEP','BARCELONA','BARCELONA','01/01/63',234567,2)

SELECT IDENT_SEED('REGIONES') AS [PRIMER VALOR]
SELECT IDENT_CURRENT('REGIONES') AS [VALOR ACTUAL]
SELECT IDENT_INCR('REGIONES') AS [INCREMENTO]
SELECT IDENTITYCOL,REGIONES FROM REGIONES
```

TABLA ALUMNOS

```
insert into alumnos
(dni,Nombre,Apellido1,Apellido2
,Fecha_nac,Localidad,Provincia,Cod_Colegio)
values
('54132456-R','Marta','Serrano','Lopez','15/03/93','MADRID','MADRID',13)
```

PARA INTRODUCIR A ESTE ALUMNO HAY QUE CAMBIAR LA COLUMNA DE LOCALIDAD PARA QUE ADMITA MAS CARACTERES.

```
alter table alumnos
alter column Localidad nvarchar(50) null

insert into alumnos
(dni,Nombre,Apellido1,Apellido2
,Fecha_nac,Localidad,Provincia,Cod_Colegio)
values
('3414356-R','Javier','Morgado','Palomo','15/03/93','ARENALES DEL
SOL','ALICANTE',11)

insert into alumnos
(dni,Nombre,Apellido1,Apellido2
,Fecha_nac,Localidad,Provincia,Cod_Colegio)
values
('3254353-R','Miguel','Torres','Tormo','15/03/95','LLOBREGAT','BARCELONA',12)
```

- Eliminar un tipo de dato que hayamos usado en las tablas.

```
exec sp_droptype Tnulo
```

No se puede eliminar el dato porque está vinculado a una tabla, para poder eliminarlo deberíamos modificar la tabla y cambiar todos los campos donde el dato está presente.

- Borrar la tabla Regiones.

drop table regiones

¿Qué ocurre?. ¿Cómo lo solucionamos?

Servidor: mensaje 3726, nivel 16, estado 1, línea 1
No se puede quitar el objeto 'regiones'. Hay una referencia a él en una restricción FOREIGN KEY.

Se soluciona quitando las dependencias que tienen la tablas.

- Borrar todas las tablas, tipos de usuario y base de datos. Explicar en que orden deben borrarse.

```
drop table alumnos
drop table profesores
drop table colegios
drop table colegios
```

```
Hay que quitar primero las relaciones de FK para luego quitar las
tablas siguientes.
```

RESTRICCIONES, VALORES POR DEFECTO Y REGLAS

1. A partir de la información que tenemos de la base de datos Hospital, crear la integridad de los datos utilizando reglas, valores por defecto..., dependiendo del análisis que hagamos previamente. Crear una nueva base de datos y las tablas Emp y Dept.

```
--Creo la base de datos
create database Emp
on primary
(Name = Emp_Data
,Filename = 'C:\Archivos de programa\Microsoft SQL
Server\MSSQL\Data\Emp_Data.mdf'
,Size = 1 MB
,MaxSize = 2MB
,Filegrowth = 2%)
Log on
(Name = Emp_Log
,Filename = 'C:\Archivos de programa\Microsoft SQL
Server\MSSQL\Data\Emp_Log.ldf'
,Size = 1 MB
,MaxSize = 2MB
,Filegrowth = 2%)

--Creo las tablas y utilizo los valores Null y Not Null convenientemente.
use Emp
go
create table Emp
(Emp_no int not null
,Apellido nvarchar(20) not null
,Oficio nvarchar(20) not null
,Dir int null
,Fecha_alt smalldatetime null
,Salario int null
,Comision int null
,Dept_no int null)
go
create table Dept
(Dept_no int not null
,Dnombre nvarchar(19) not null
,Loc nvarchar(20) null)

--Poner Restricción Primary Key en Departamento
Alter Table Dept
add constraint PK_Dept Primary Key(Dept_no)

--Poner Primary Key en Empleados
Alter Table Emp
add constraint Pk_Emp Primary Key(Emp_no)
```

```
--Poner la Foreign Key en Empleados
Alter Table Emp
add constraint FK_Emp Foreign Key (Dept_no)
references Dept (Dept_no)
on Update Cascade
on Delete Cascade

--Creo una restricción Check para el Salario
Alter Table Emp
add constraint CK_Salario
Check (Salario > 0)

--Creo una restricción Check para la Comisión
Alter Table Emp
add constraint CK_Comisión
Check (Comision >= 0)

--Creo una restricción Check para la Fecha
Alter Table Emp
add constraint CK_Fecha
Check (Fecha_alt > '01/01/1970' and Fecha_alt <=GetDate())

--Creo un Valor Predeterminado para Fecha
alter table Emp
add constraint DF_Fecha
Default GetDate() for Fecha_alt

--Creo un Valor Predeterminado Comisión
Alter Table Emp
add constraint DF_Comision
Default 0 for Comision

--Creo una restricción Default para la localidad en Departamento
Alter Table Dept
add constraint DF_Loc
Default 'DESCONOCIDA' for Loc

--Creo una regla para que todos los empleados tengan asociado director
Create Rule Director_Rule as
@Director in(7566,7698,7782,7839)
GO
exec sp_bindrule Director_Rule,'Emp.Dir'
exec sp_unbindrule 'Emp.Dir',Director_Rule

--Creo una regla para Oficio que mantenga la integridad
Create Rule Oficio_Rule as
@Oficio in('ANALISTA','EMPLEADO','PRESIDENTE','DIRECTOR')
GO
exec sp_bindrule Oficio_Rule,'Emp.Oficio'
exec sp_unbindrule 'Emp.Oficio',Oficio_Rule
```

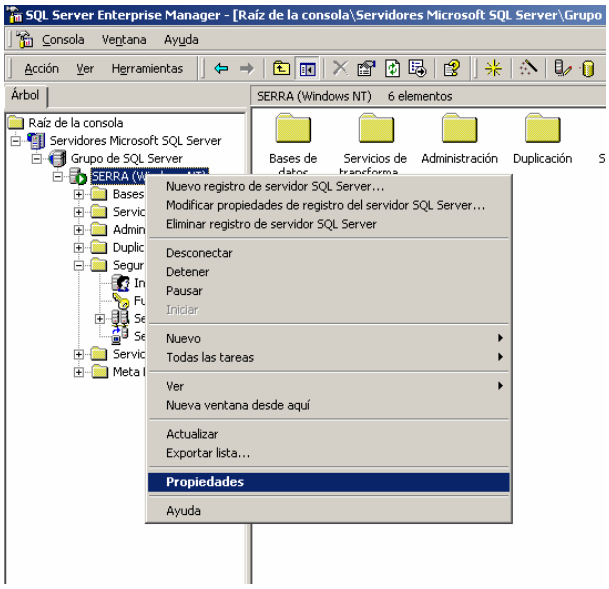
2. Insertar los datos convenientemente para verificar la integridad de nuestra base de datos.

Debo insertar primero al Director y un departamento para poder comenzar a utilizar los datos, lo debo introducir antes de crear las reglas.

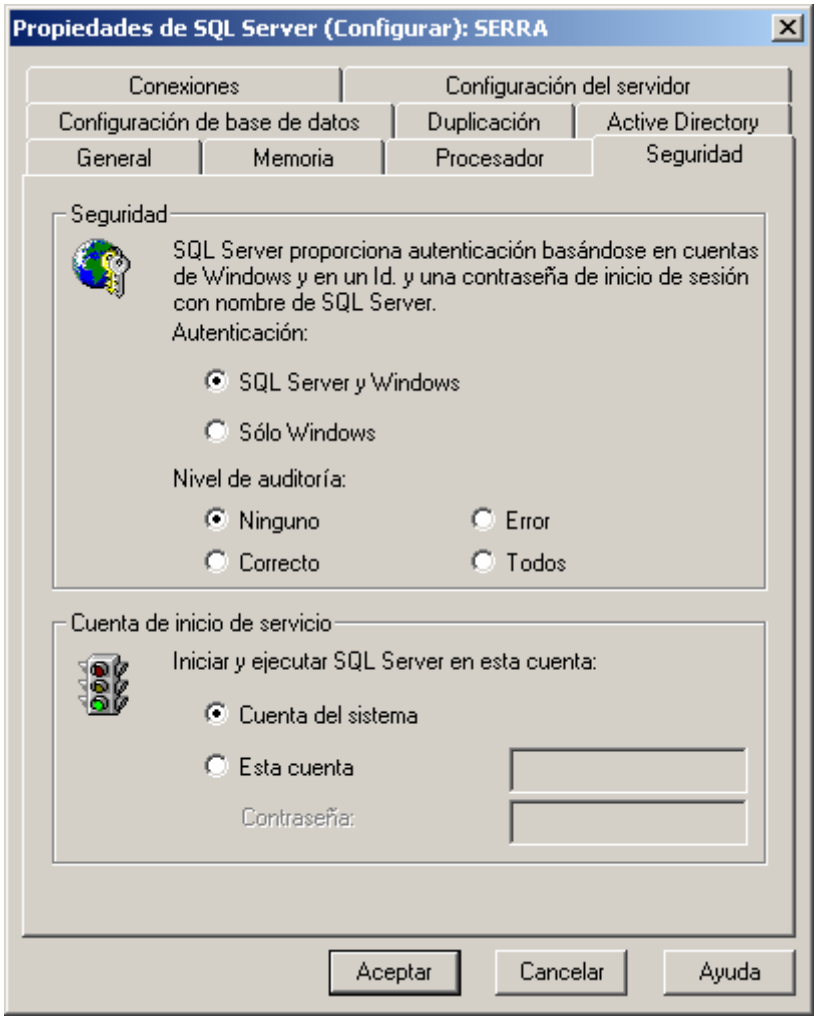
```
insert into dept select * from hospital.dbo.dept where dept_no = 10
insert into emp select top 1
emp_no,apellido,oficio,dir,fecha_alt,salario,comision,dept_no
from hospital.dbo.emp where oficio = 'Presidente'
```

Las demás inserciones debo hacerlas jerarquicamente con todas las reglas y restricciones activadas, es decir, primero los directores y luego los demás empleados.

PERMISOS Y USUARIOS



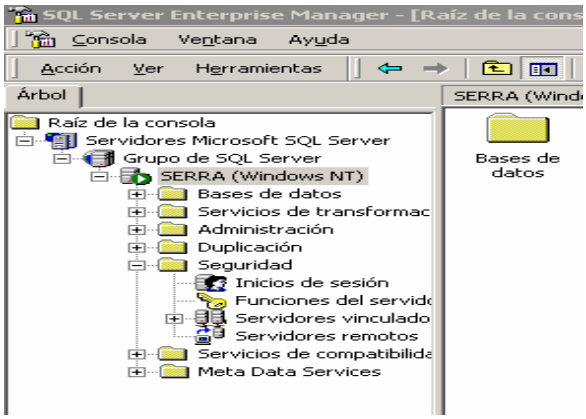
Lo primero que hay que hacer es verificar la seguridad que tenemos en SQL Server. Para ello pulsamos sobre nuestro servidor y seleccionamos las propiedades, después pulsamos en la pestaña de Seguridad. Necesitamos tener la seguridad mixta, SQL Server y Windows para poder crear usuarios y contraseñas dentro de SQL.



Vamos a crear un nuevo usuario:

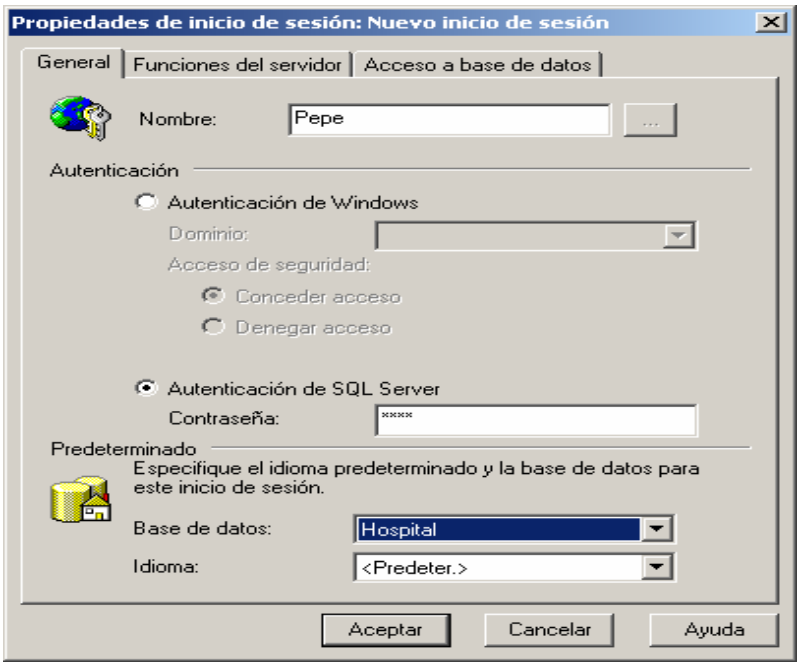
- Usuario: Pepe
- Contraseña: Pepe
- Base de datos Predeterminada: Empleados

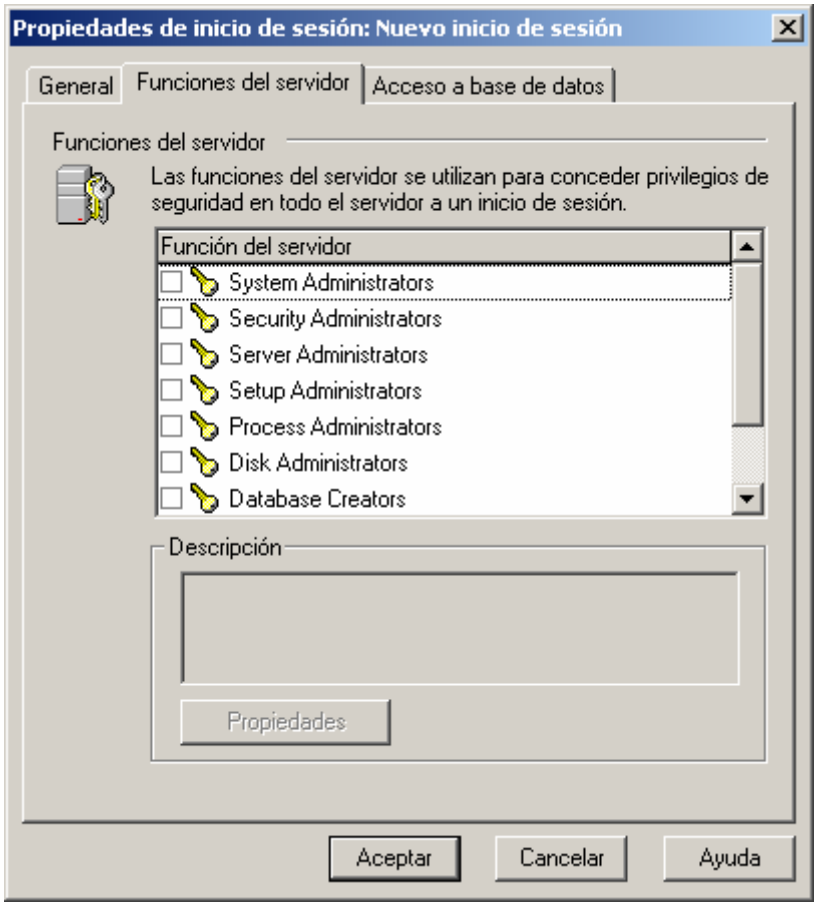
Ir a la carpeta de seguridad del servidor



Tenemos los inicios de sesion, Funciones predefinidas por el sistema y tienen alcance sobre toda las bases de datos del servidor.
Por ejemplo, Database Creator ofrece permisos sobre crear nuevas bases de datos en el servidor.

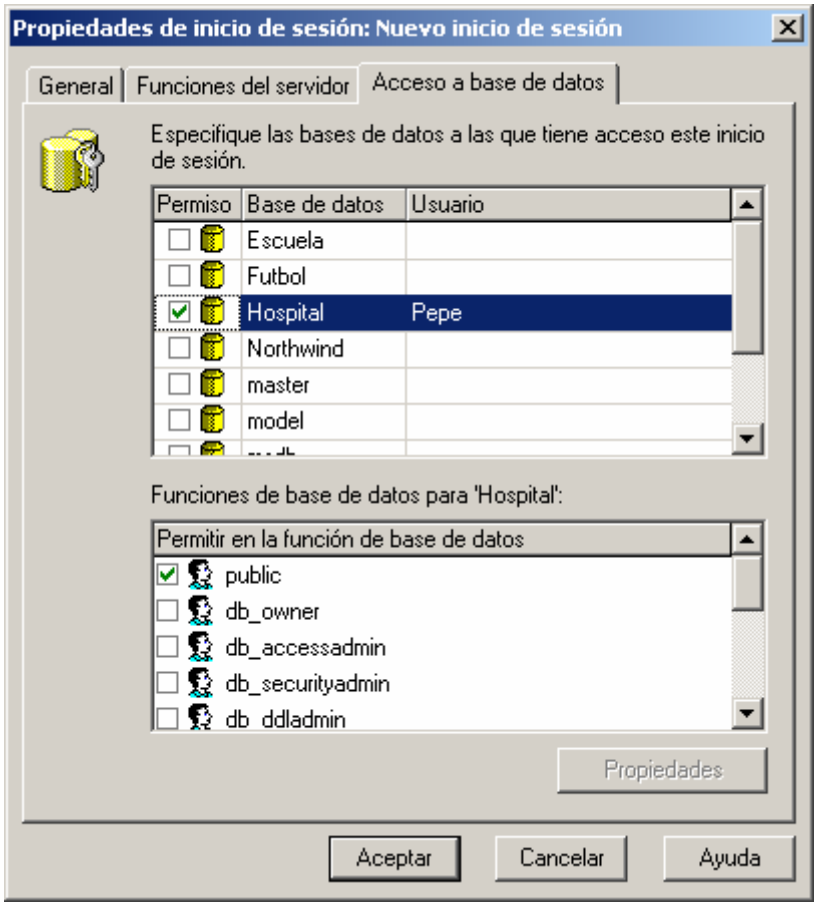
Vamos a crear un usuario: Inicio de sesion, boton derecho y Nuevo Inicio de Sesión.
Debemos decidir como se valida el usuario, por defecto está por Windows, vamos a validar por SQL ya que no tenemos permiso para administrar usuarios en Windows.



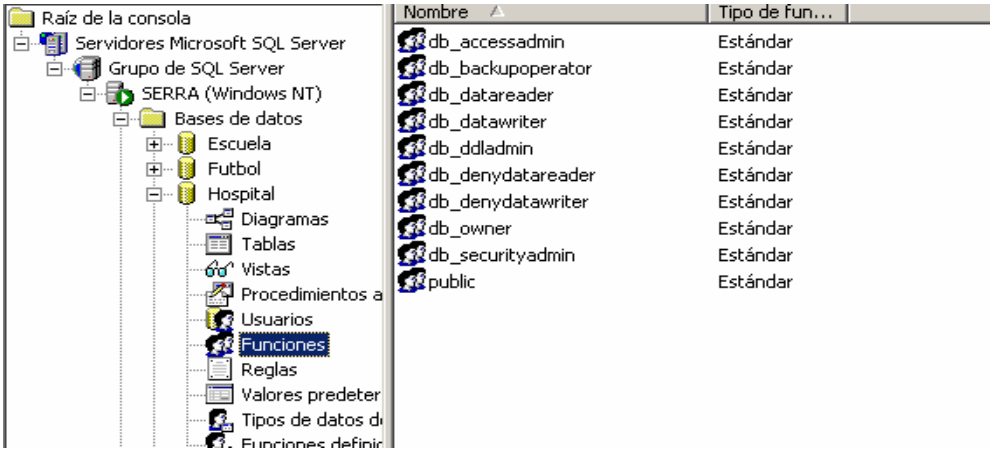


Funciones del Servidor
Poderes para los usuarios, sus funciones y sus capacidades, privilegios sobre todas las bases o el servidor. Nos muestra una pequeña descripción de cada una al pulsar sobre ellas.

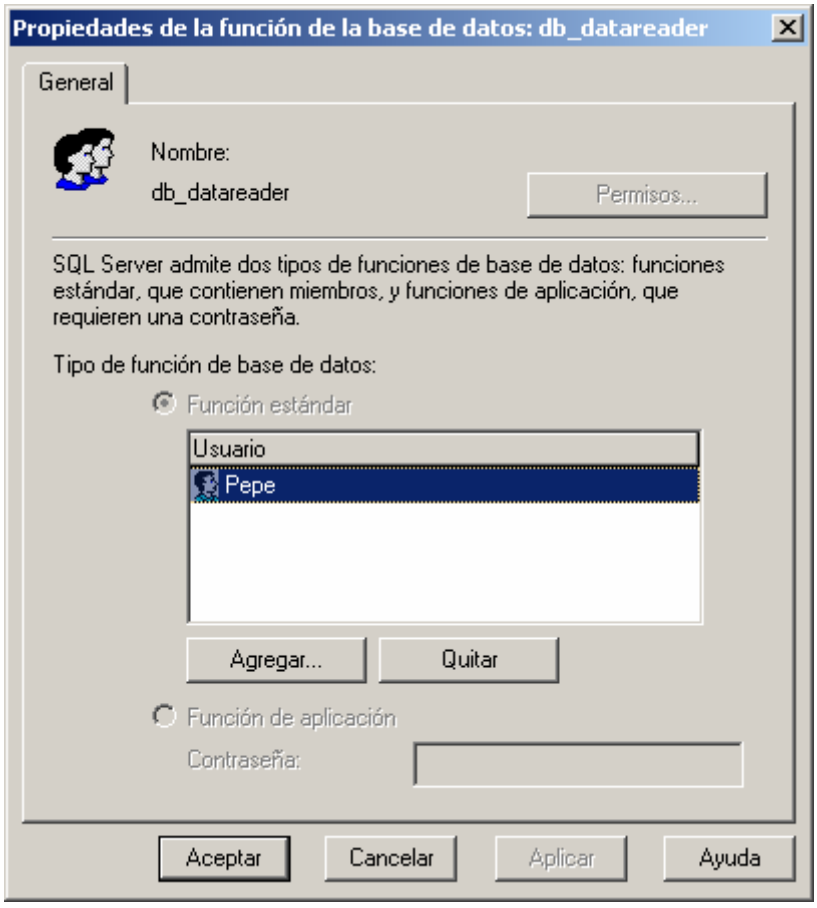
Seleccionamos una base de datos y en la parte inferiror me aparecen funciones predefinidas que tienen alcance para la propia base de datos.
Existen diferentes tareas, escribir, dar y quitar permisos a ususarios...
Por defecto sale Public, que son los permisos minimos que tiene un usuario sobre la base de datos y no se pueden quitar.
Puedo crear grupos que tengan ya contenidos los propios permisos para cada uno de los grupos en la base de datos.
Db_datareader: Solo ofrece permiso de lectura
Db_datawriter: Permite escribir(Update, Insert y Delete)



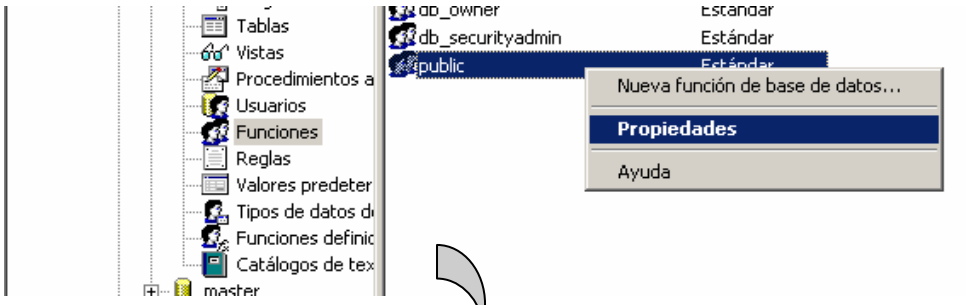
Le damos al usuario poderes sobre la función Db_datareader sobre la base de datos Hospital.
Para ver los permisos nos vamos a la base de datos Hospital, seleccionamos Funciones.

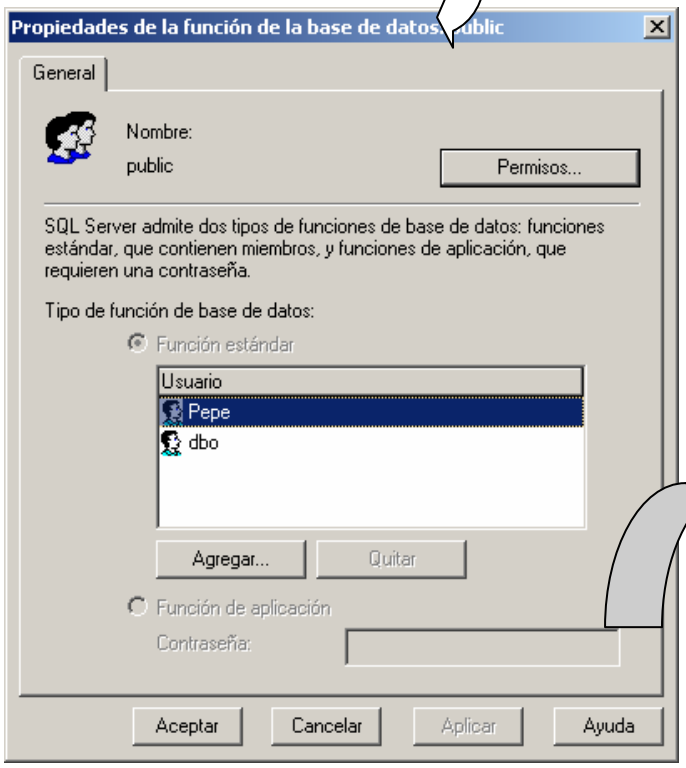


Elegimos db_datareader, botón derecho, propiedades y aparecen los usuarios vinculados a esta función.

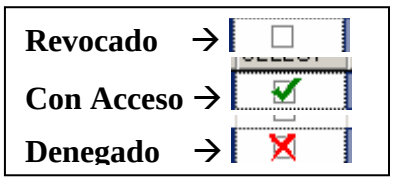


Vamos a ver los permisos que tiene la función Public:

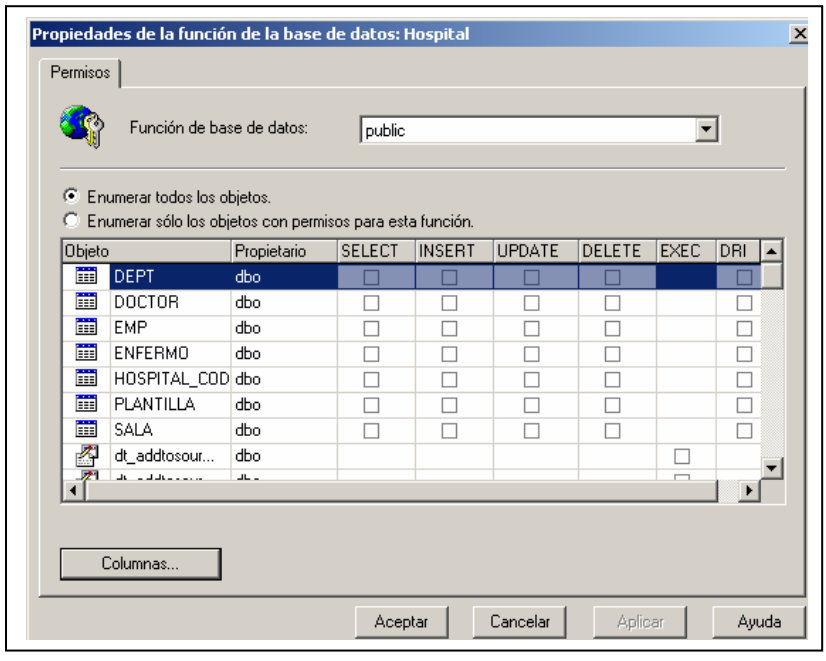




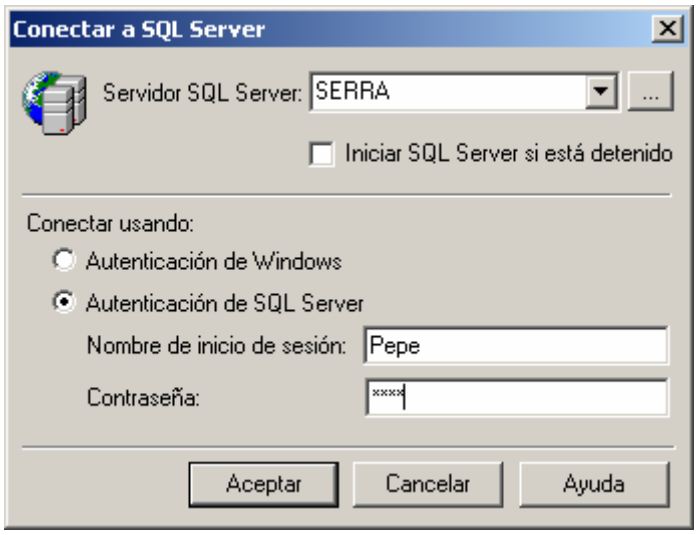
Seleccionamos al usuario y pulsamos en los permisos



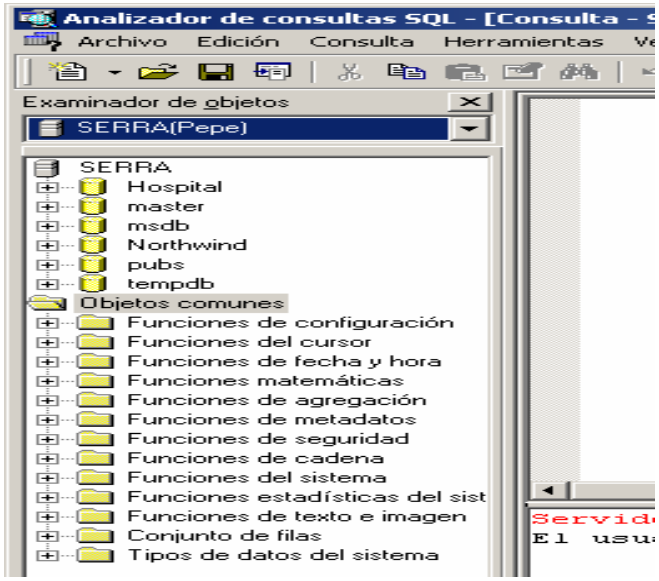
Se imponen los permisos al usuario antes que las funciones. Con esto ya tenemos a nuestro usuario creado y con los permisos que hemos querido concederle.



Ahora vamos a comprobar los permisos en el analizador de consultas.
Desde el administrador corporativo: Herramientas, Analizador de Consultas y Conectar.



Escribimos el nombre de nuestro nuevo usuario y su contraseña para poder concectar.



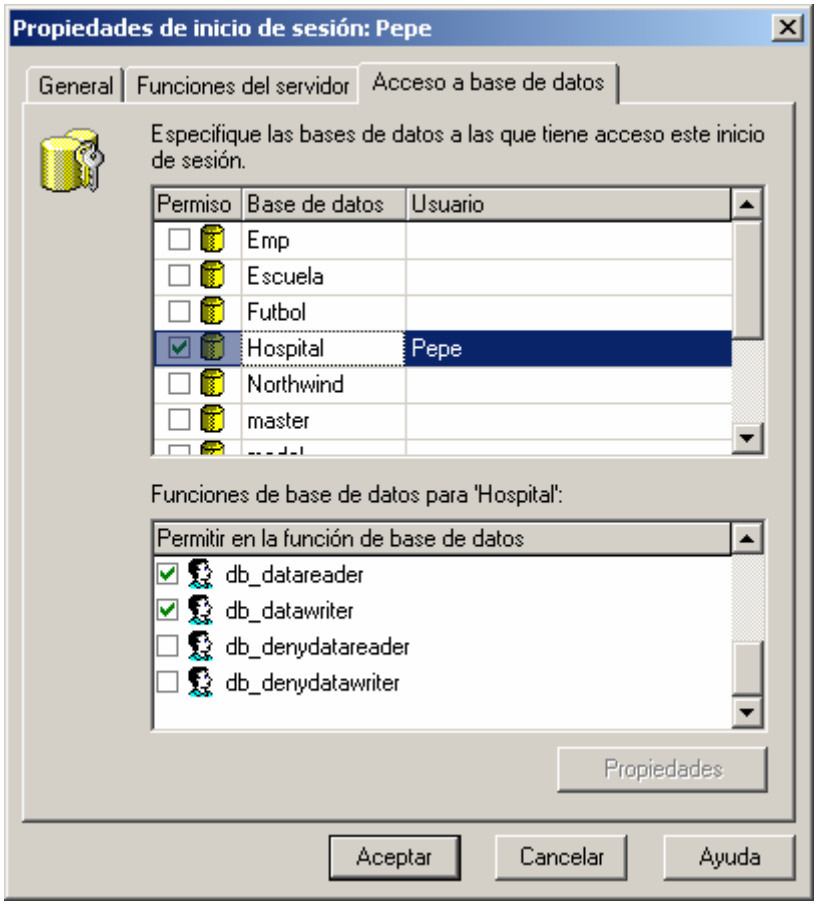
Solo permitirá acceso a las tablas que le hemos dado permiso y a las de ejemplo que vienen con SQL Server.

Si intentamos hacer un Insert en la tabla:
Insert into dept (Dept_no, DNombre, Loc) values (50,'EDICION','ELCHE')
Servidor: mensaje 229, nivel 14, estado 5, línea 1
Permiso INSERT denegado para el objeto 'DEPT', base de datos 'Hospital', propietario 'dbo'.

Intentar hacer un Select, en este caso me permite realizar la consulta, porque tengo permisos de tipo Select.
Select * from Dept

	DEPT_NO	DNOMBRE	LOC
1	10	CONTABILIDAD	SEVILLA
2	20	INVESTIGACION	MADRID
3	30	VENTAS	BARCELONA
4	40	PRODUCCION	BILBAO

Conceder más permisos al mismo usuario:



Ahora vamos a conceder más permisos al usuario Pepe, lo vamos a incluir en la función db_datawriter para que además pueda realizar consultas de acción sobre la tabla (Insert, Update y Delete)

Sin necesidad de volver a conectar con el analizador de consultas ya tendríamos los permisos para insertar, borrar o actualizar:

```
Insert into dept (Dept_no, DNombre, Loc) values (50,'EDICION','ELCHE')
```

(1 filas afectadas)

Los Permisos Select siguen igual y se mantienen.

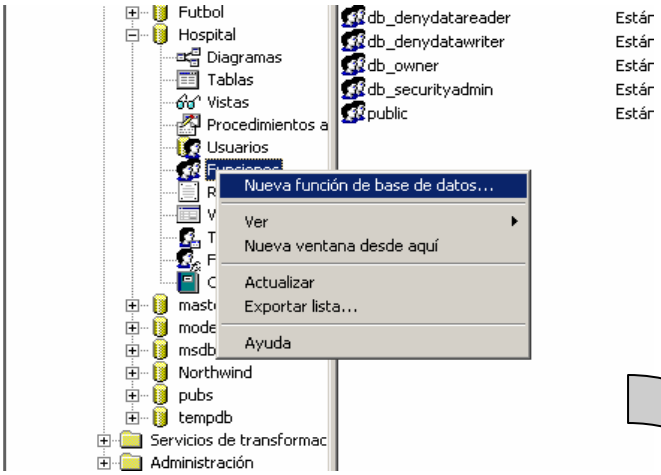
```
Select * from Dept
```

	DEPT_NO	DNOMBRE	LOC
1	10	CONTABILIDAD	SEVILLA
2	20	INVESTIGACION	MADRID
3	30	VENTAS	BARCELONA
4	40	PRODUCCION	BILBAO
5	50	EDICION	ELCHE

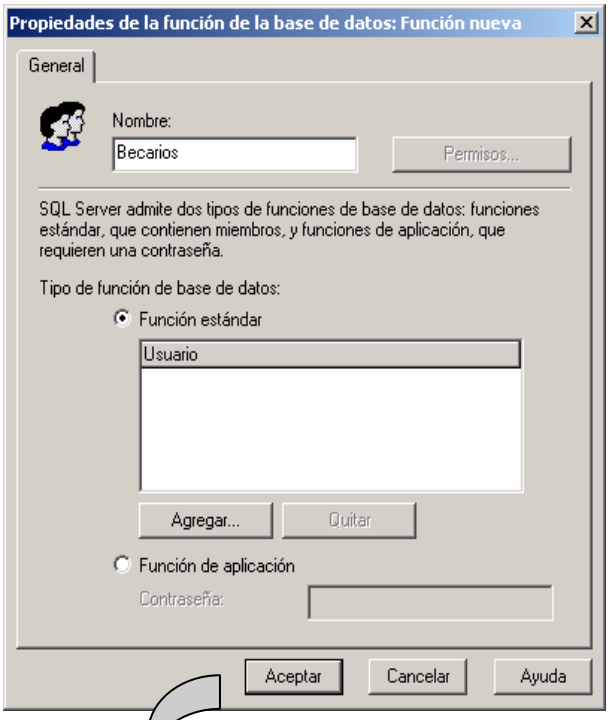
```
delete from dept where Loc = 'ELCHE'
```

(1 filas afectadas)

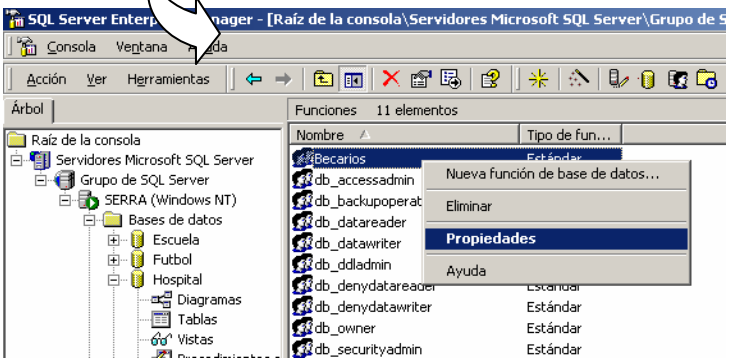
Creación de un grupo de funciones para un grupo de usuarios con las mismas características.



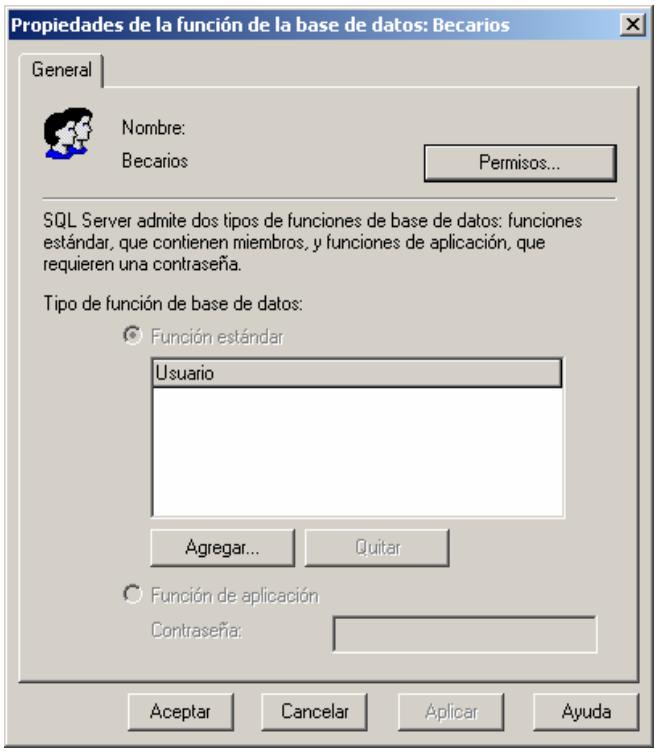
Seleccionamos una tabla, pulsamos sobre Funciones con el botón derecho y seleccionamos: Nueva función de base de datos.



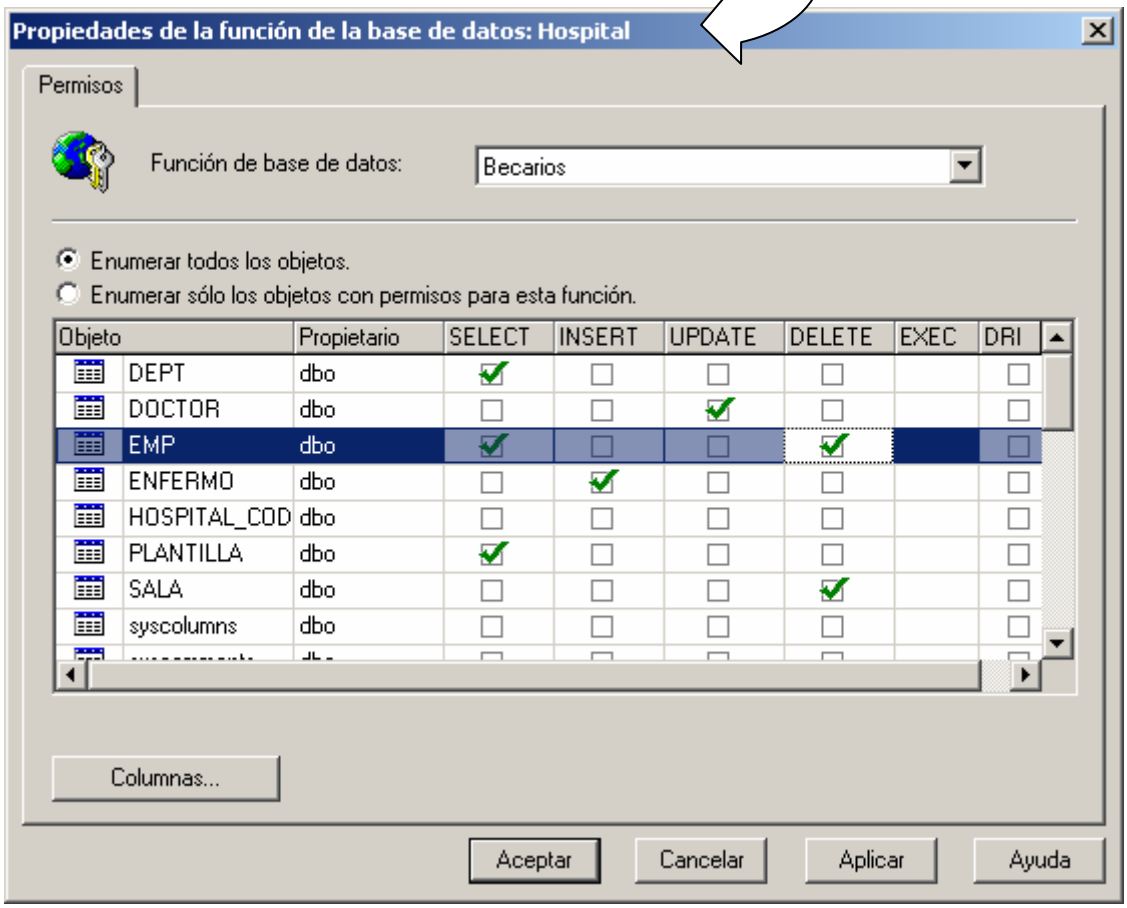
Escribimos el nombre de nuestra nueva Función, no le damos todavía los usuarios ni los permisos, esto es solamente para crearla.



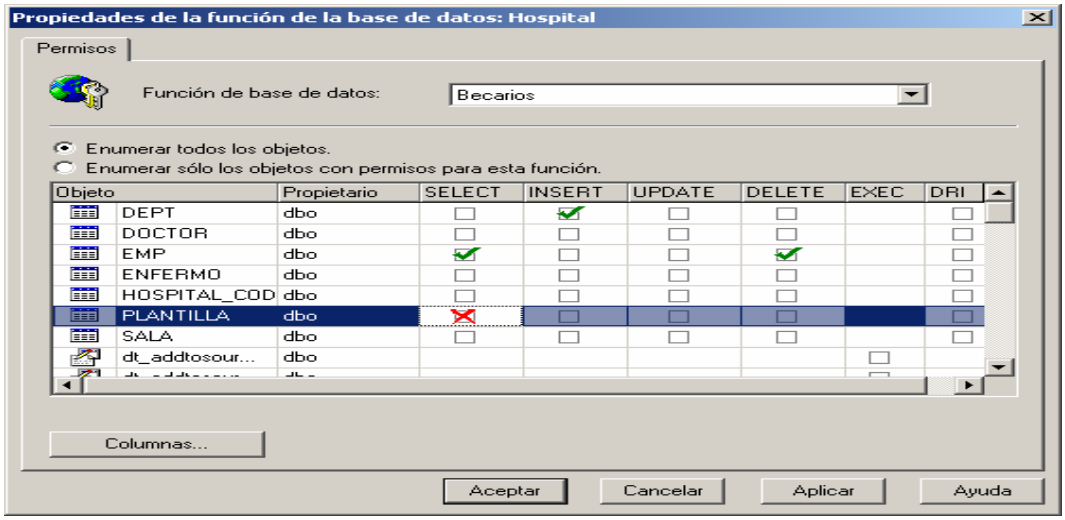
Una vez creada la función, la seleccionamos con el botón derecho y pulsamos sobre sus propiedades.



Una vez aquí y seleccionando los permisos, podremos otorgar permisos de escritura, selección o lo que queramos a ese grupo de usuarios en la función. Después podremos añadir los usuarios que queramos que estén dentro de esa función.



Los permisos que tiene el usuario siguen actuando, y además hereda los permisos que tiene de la Función Becarios. Los permisos siguen funcionando a no ser que se le deniegue explicitamente el acceso a un permiso sobre la función, entonces el usuario no puede ejecutar la orden si está dentro de la función.

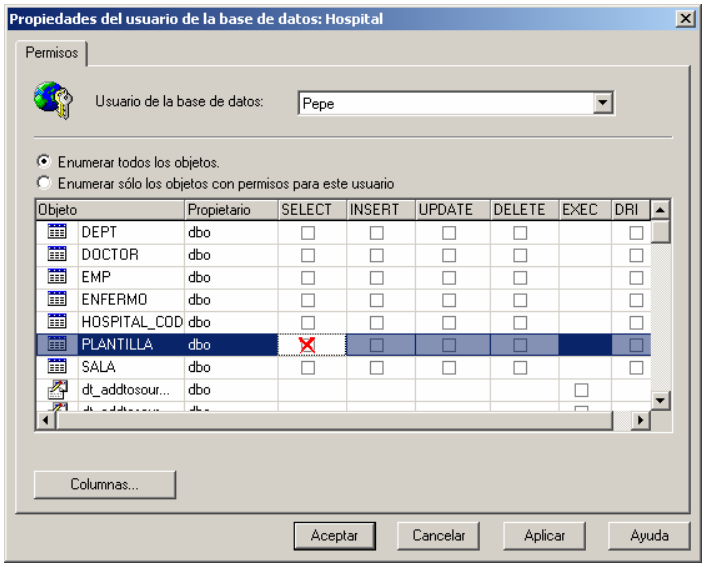


En este caso el usuario PEPE tiene los permisos de Select y consultas de acción sobre la base de datos Hospital. Pero el usuario está dentro de la función Becarios y le hemos denegado el acceso a realizar un Select sobre la tabla Plantilla.
Al realizar el Select sobre la tabla plantilla tiene los permisos denegados por ser miembro de la función Becarios. Se le niega el acceso expresamente sobre acciones en una tabla y no importa los permisos que tenga el propio usuario.

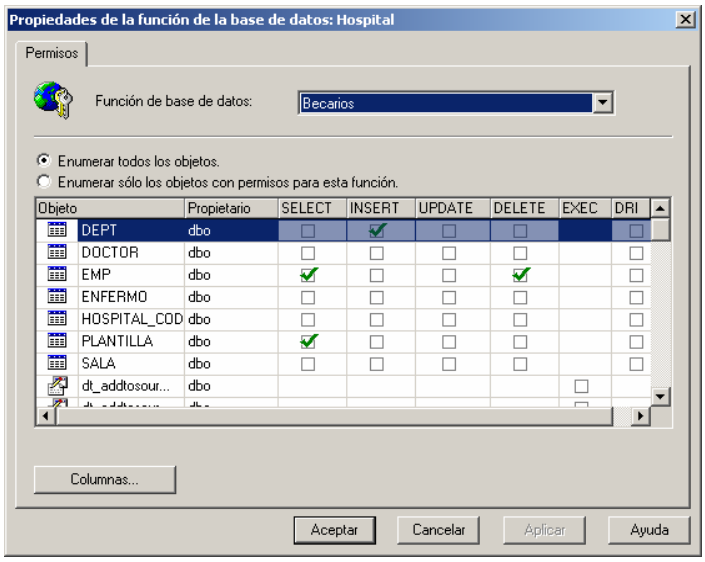
```
select * from plantilla
Servidor: mensaje 229, nivel 14, estado 5, línea 1
Permiso SELECT denegado para el objeto 'PLANTILLA', base de datos 'Hospital',
propietario 'dbo'.
```

Si denegamos el permiso Select sobre la tabla Plantilla en el propio usuario, prevalece el usuario sobre la función, las clausulas que deniegan acceso a la base de datos prevalecen sobre las que dan permiso.

USUARIO PEPE



FUNCION BECARIOS




```
select * from plantilla
Servidor: mensaje 229, nivel 14, estado 5, línea 1
Permiso SELECT denegado para el objeto 'PLANTILLA', base de datos 'Hospital',
propietario 'dbo'.
```

SEGURIDAD

Para crear un usuario, debemos realizar los siguientes pasos:

- Crear el usuario asignándole una cuenta de inicio de sesión.
- Asignar el usuario a una base de datos y darle permiso de acceso a ella
- Asociar el usuario a una función de usuario la cuál contiene unos determinados permisos para acceder a la base de datos.

❑ Cuentas de inicio de sesión

Cuando un usuario se conecta a un servidor, lo hace a través de una cuenta facilitada por el administrador de la base de datos. Esta cuenta tiene una serie de permisos que el administrador da, estos permisos pueden ser de restricción a determinadas tablas, o por ejemplo que un usuario pueda solo insertar, o seleccionar en determinadas tablas.

Existen dos formas de entrar a la base de datos, bien mediante el usuario de windows nt ó 2000, ó mediante el usuario de la base de datos.
Mediante el usuario de windows, facilitamos al equipo desde el que nos conectamos, los recursos del servidor de sql server, pudiendo incluso bloquear el equipo desde el que se conecta el usuario.

❑ Acceso a una base de datos

Para que un usuario tenga acceso a una base de datos, después de crearle una cuenta de inicio de sesión, y asociársela a una base de datos, hemos de darle permisos para que pueda acceder a esa base de datos. Posteriormente, mediante funciones le indicaremos que es lo que puede hacer en esa base de datos.
Puede tener permisos para una base de datos o para todo el servidor.

❑ Funciones

Agrupamos usuarios y sobre estos usuarios daremos permisos a ese grupo de usuarios.
Las funciones de usuario, se usan para establecer los permisos que un grupo de usuario tendrá sobre una determinada base de datos.
Una vez creado el usuario, y asignada la base de datos, podemos asociar este usuario a la función que queramos para controlar los permisos que tiene sobre la base de datos asignada.

Hay una serie de funciones y usuarios predefinidos en la base de datos.

❑ Crear un usuario

Para crear un usuario y poder usarlo, hemos de establecer todos los pasos descritos en los puntos anteriores.
Estos pasos, los realizaremos todos en el analizador de consultas:

1. Creamos el inicio de sesión del usuario.

   - **SP_ADDLOGGIN**
     Crea una cuenta de inicio para un usuario y lo asigna a una base de datos.
     Sintaxis:

     SP_ADDLOGIN ‘Usuario’, ‘Contraseña’, ‘Base de datos’

     EXEC SP_ADDLOGIN 'Pepe', 'Pepe','Hospital'
     Creado nuevo inicio de sesión.

   - **SP_DROPLOGIN**
     Borra una cuenta de inicio de un usuario, siempre y cuando el usuario no esté conectado y no tenga permisos sobre ninguna base de datos (si los tiene hemos de revocarlos antes de borrar la cuenta de inicio y el usuario)Sintaxis:

     SP_DROPLOGIN 'Usuario'

     EXEC SP_DROPLOGIN 'Pepe'
     Inicio de sesión quitado.

     Nota: Si el usuario tiene permisos asignados sobre una base de datos, debemos quitar primero esos permisos antes de borrarlo, ya que sino dará error al intentarlo.

2. Asignamos permisos al usuario sobre la base de datos que queramos

   - **SP_GRANTDBACCESS**

3. Para ello usaremos el procedimiento almacenado de sistema SP_GRANTDBACCESS.
   Sintaxis:
   SP_GRANTDBACCESS ‘Usuario’

   Use Hospital
   GO
   SP_GRANTDBACCESS 'Pepe'
   Concedido a la base de datos acceso a 'Pepe'.

Si queremos revocar estos permisos al usuario, usaremos el procedimiento almacenado de sistema SP_REVOKEDBACCESS. Sintaxis:
SP_REVOKEDBACCESS ‘Usuario’
Es decir en este caso sería:
EXEC SP_REVOKEDBACCESS 'Pepe'
El usuario se ha quitado de la base de datos actual.

4. Probamos el usuario creado entrando en el analizador de consulta con su login y password

Una vez en el analizador de consultas, solo tendremos acceso a la base de datos asociada al usuario y las bases de datos de ejemplo.

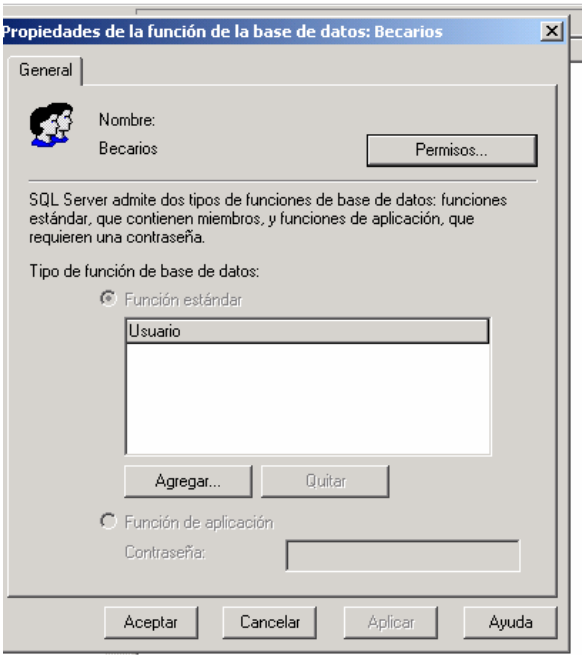
5. Asociar el usuario a la función que queramos.

Crear Funciones (grupos de usuarios) y establecer sus permisos

Para ello crearemos la función mediante el procedimiento almacenado de sistema SP_ADDROLE

```
EXEC SP_ADDROLE 'Becarios'
```

Agregada la nueva función.



Una vez realizado esto, vemos que en el apartado “Funciones” de la base de datos, está la función creada, si hacemos doble click sobre ella, vemos que no tiene usuarios asignados:

Para establecer los permisos de la función que hemos creado, usaremos el comando GRANT. Sintaxis:

GRANT Permisos
ON Tabla / Objeto
To Función / Usuarios

Es decir en este caso sería:

```
GRANT INSERT, UPDATE, DELETE
ON Emp
To Becarios
```

También podemos establecer permisos directamente sobre los usuarios, poniendo el nombre de estos en lugar de la función.

```
GRANT INSERT, UPDATE, DELETE
ON Emp
To Pepe, Pepa
```

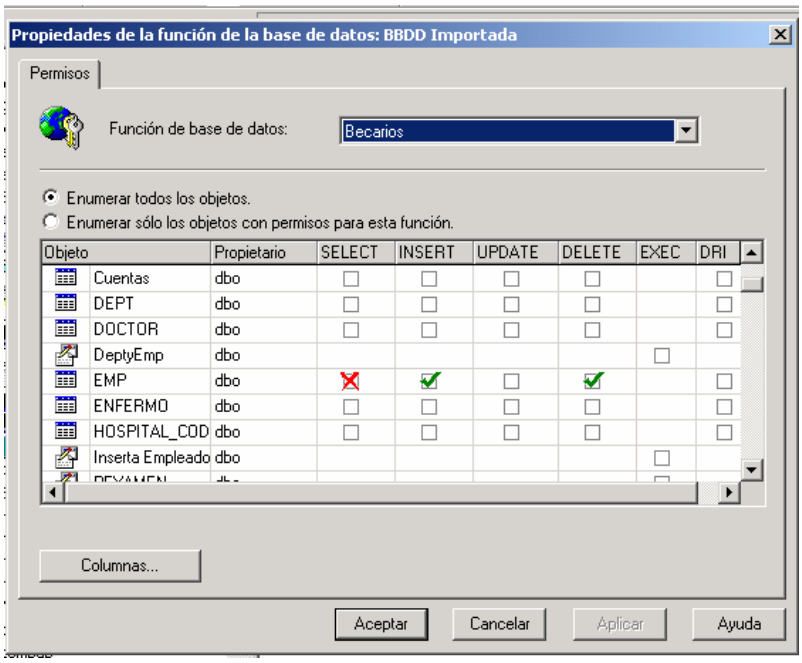
Para denegar permisos sería mediante el comando DENY.
Sintaxis:
DENY Permisos
On Tabla
To Función

```
DENY SELECT
ON Emp
TO Becarios
```

Para revocar permisos ya asignados mediante GRANT, usaremos el comando REVOKE

REVOKE Permisos
On Tabla
To Función

```
REVOKE UPDATE
On Emp
to Becarios
```



Si después de ejecutar todos estos comandos hacemos doble click sobre la función y vemos los permisos.

Asociar usuarios a una Función.

Para asociar un usuario a una función usaremos el procedimiento almacenado de sistema SP_ADDROLEMEBER. Sintaxis:
SP_ADDROLEMEMBER ‘Función’, ‘Usuario’

Es decir, en este caso sería:

```
EXEC SP_ADDROLEMEMBER 'Becarios', 'Pepe'
'Pepe' agregado a la función 'Becarios'.
```

Una vez realizados estos pasos, si entramos en el analizador de consultas con el usuario Pepe, si intentamos realizar una consulta no permitida, nos advertirá mediante un mensaje que no podemos realizar la consulta:

```
Select * from emp

Servidor: mensaje 229, nivel 14, estado 5, línea 1
Permiso SELECT denegado para el objeto 'EMP', base de datos 'Hospital', propietario 'dbo'.
```

Para eliminar un usuario de una función, usaremos el procedimiento almacenado de sistema SP_DROPROLEMEMBER. Sintaxis:
SP_DROPROLEMEMBER ‘Función’, ‘Usuario’
Es decir en este caso sería:

```
EXEC SP_DROPROLEMEMBER 'Becarios', 'Pepe'
'Pepe' quitado de la función 'Becarios'.
```

Una vez tengamos la función sin usuarios asignados, podremos borrarla, para borrarla usaremos el procedimiento almacenado de sistema SP_DROPROLE. Sintaxis:
SP_DROPROLE ‘Función’
En este caso sería:

```
EXEC SP_DROPROLE ‘Becarios’
Función quitada.
```

SP_ADDSRVROLEMEMBER Añade a una función de sistema el usuario que queramos.
Sintaxis: SP_ADDSRVROLEMEMBER 'Usuario','Funcion'

Damos permiso al usuario para modificar y crear bases de datos.

```
EXEC SP_ADDSRVROLEMEMBER 'Pepe','DbCreator'
SP_ADDSRVROLEMEMBER 'PEPE','DBCREATOR'
```

SP_DROPSRVROLEMEMBER Quita a un usuario de la función de sistema usuario que seleccionemos.
Sintaxis: SP_ADDSRVROLEMEMBER 'Usuario','Funcion'
Quitamos de la función al usuario ‘Pepe’

```
EXEC SP_ADDSRVROLEMEMBER 'Pepe','DbCreator'
'PEPE' quitado de la función 'DBCREATOR'.
```

Otro método que tenemos para poder ofrecer permisos es con la opción
With Grant Option

Con esta opción después de la sentencia de conceder derechos, permito al usuario al que estoy concediendo permisos pueda conceder permisos a su vez sobre los privilegios que se le han otorgado.

Sintaxis:
Grant Select, Insert, Update, Delete
On Tabla / Vista
To Usuario / Funcion
With Grant Option

Ejemplo:
Un administrador concede permisos a Pepe para poder hacer selecciones sobre la tabla Emp.
Usuario → Administrador
Grant Select
On Emp
To Pepe
With Grant Option

Este usuario podrá a su vez conceder permisos a otro usuario sobre sus privilegios, es decir, sobre la tabla emp y solamente con Select.

Usuario → Pepe

Grant Select
On Emp
To Luisa

La cadena termina aquí, ya que Pepe no ha concedido permisos a Luisa para que pueda conceder permisos a su vez.
Si el administrador revocase los permisos Select a Pepe, estos permisos se revocan a su vez sobre el usuario Luisa, ya que no tiene permisos de nadie más para ver la Tabla.
Otra opción es que el usuario Pepe conceda permisos a Luisa pero solamente sobre unos determinados campos.

Grant Select (Emp_no, Apellido, Fecha_alt, Dept_no)
On Emp
To Luisa

Esto se puede utilizar también pero sería más conveniente utilizar una Vista.

MODULO 8: IMPLEMENTACIÓN DE VISTAS

Las vistas son consultas ya realizadas, pueden ser de una sola tabla o de varias. Muy útiles si por ejemplo queremos que un usuario solo tenga acceso a unas determinadas columnas de una tabla pero no a otras, para eso crearemos una vista con las columnas que puede ver. Las vistas tienen la propiedad que si cambiamos el nombre de las tablas o columnas a las que hace referencia, automáticamente se cambian en esta.

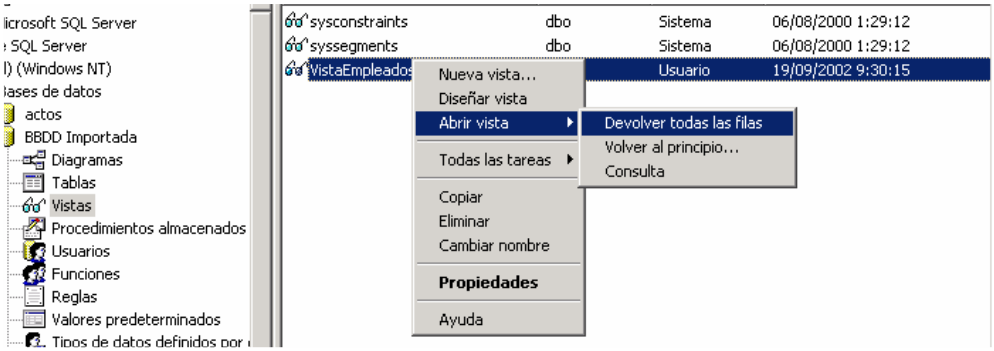
- **Create View**

Crea una vista. Sintaxis:
Create View NombreVista
As
Sentencia Sql

Ejemplo:

```
CREATE VIEW VistaEmpleados
As
Select Emp_no, Apellido, Fecha_Alt, Dnombre
From Emp
Inner Join Dept
On
Emp.Dept_No = Dept.Dept_no
```

Crea una vista llamada VistaEmpleados que contiene nº de empleado, apellido, fecha de alta, y nombre de departamento de las tablas Empleados y Departamento.



Vemos que la vista aparece en el apartado 'Vistas' del administrador corporativo(si no aparece pulsamos F5 para actualizar).Si recuperamos todas sus filas...

Emp_no	Apellido	Fecha_Alt	Dnombre
7499	ARROYO	22/02/1987	VENTAS
7521	SALA	22/02/1980	VENTAS
7654	MARTIN	28/09/1981	VENTAS
7698	NEGRO	01/05/1984	VENTAS
7782	CEREZO	09/06/1978	CONTABILIDAD
7839	REY	17/11/1975	CONTABILIDAD
7844	TOVAR	08/09/1981	VENTAS
7900	JIMENO	03/12/1990	VENTAS

Si después de esto, modificamos los datos de las tablas aceptadas por la vista...

```
Update Dept Set Dnombre = 'Cuentas'
where Dnombre = 'Contabilidad'
```

Volvemos a ver los datos de la vista y vemos que también han cambiado, ya que la Vista no es una copia de los datos si no una consulta realizada sobre los datos originales...

Emp_no	Apellido	Fecha_Alt	Dnombre
7499	ARROYO	22/02/1987	VENTAS
7521	SALA	22/02/1980	VENTAS
7654	MARTIN	28/09/1981	VENTAS
7698	NEGRO	01/05/1984	VENTAS
7782	CEREZO	09/06/1978	Cuentas
7839	REY	17/11/1975	Cuentas
7844	TOVAR	08/09/1981	VENTAS
7900	JIMENO	03/12/1990	VENTAS

También se pueden realizar consultas usando campos de tablas mezclados con campos de vistas.

Ejemplo:

```
Select Emp_No, Apellido, Dnombre
From VistaEmpleados
Union
Select Empleado_no, Apellido, Hospital_Cod
From Plantilla
```

	Emp_No	Apellido	Dnombre
1	1	1	22
2	1009	Higueras D.	22
3	1280	Amigo R.	45
4	3106	Hernández J.	13
5	3754	Díaz B.	13
6	6065	Rivera G.	22
7	6357	Karplus W.	18
8	7379	Carlos R.	22
9	7499	ARROYO	VENTAS
10	7521	SALA	VENTAS
11	7654	MARTIN	VENTAS
12	7698	NEGRO	VENTAS
13	7782	CEREZO	Cuentas
14	7839	REY	Cuentas
15	7844	TOVAR	VENTAS
16	7900	JIMENO	VENTAS
17	8422	Bocina G.	22

Cuadrículas | Mensajes

También se puede realizar una vista con datos de una consulta de una tabla y una vista.

```
Create View VistaEmpUnionHospital
As
Select Emp_no, Apellido, Dnombre
From VistaEmpleados
Union
Select Empleado_no, Apellido, Hospital_Cod
From Plantilla
```

Emp_no	Apellido	Dnombre
1	1	22
1009	Higueras D.	22
1280	Amigo R.	45
3106	Hernández J.	13
3754	Díaz B.	13
6065	Rivera G.	22
6357	Karplus W.	18
7379	Carlos R.	22
7499	ARROYO	VENTAS
7521	SALA	VENTAS
7654	MARTIN	VENTAS
7698	NEGRO	VENTAS
7782	CEREZO	Cuentas
7839	REY	Cuentas
7844	TOVAR	VENTAS
7900	JIMENO	VENTAS
8422	Bocina G.	22
8526	Frank H.	45
9901	Núñez C.	22
11222	FF	22
12121	Rey	<NULL>

Si vamos a vistas y seleccionamos todas sus filas vemos que estan los datos de la vista creada.

- **Drop View**

Borra una Vista. Sintaxis:
Drop View NombreVista
Ejemplo:

```
Drop View todos
```

- **SP_HelpText**

Muestra la consulta que realiza la vista. Sintaxis:
SP_HelpText Vista
Ejemplo

```
SP_HelpText VistaEmpleados
```

- **SP_Depends**

Muestra las tablas, campos e incluso vistas de las que depende una vista. Sintaxis:
SP_Depends Vista
Ejemplo:

```
SP_Depends VistaEmpUnionHospital
```

	name	type	updated	selected	column
1	dbo.PLANTILLA	user table	no	no	HOSPITAL_COD
2	dbo.PLANTILLA	user table	no	no	EMPLEADO_NO
3	dbo.PLANTILLA	user table	no	no	APELLIDO
4	dbo.VistaEmpleados	view	no	no	Emp_no
5	dbo.VistaEmpleados	view	no	no	Apellido
6	dbo.VistaEmpleados	view	no	no	Dnombre

- **Create View With Encryption**

Crea una Vista encriptada, de forma que si hacemos una consulta con Sp_HelpText o SP_Depends solo nos devulve un mensaje indicando que la vista está encriptada.
Sintaxis:

```
Create View NombreVista
With Encryption
As
Sentencias Sql
```

Ejemplo:

```
Create View apellidos
With encryption
as
select apellido from emp
union
select apellido from plantilla
union
select apellido from doctor

sp_helptext apellidos
```
Los comentarios de objeto han sido cifrados.

- **Create View with Check Option**

Crea una vista cuyos datos a los que hace referencia, no se pueden modificar o borrar, para que los resultados de la vista no se vean afectados. Esto es muy práctico en caso de que por ejemplo hagamos una vista con una consulta que usa where.

Sintaxis:
Create View NombreVista
As
Sentencia Sql
With Check Option

Ejemplo:

```
create view vendedores
as
select * from emp where oficio='ventas'
With Check Option
```

Con esta opción no podremos actualizar los campos que tenemos en el Where de la Vista, pero la tabla si que podemos modificarla, con lo cual ofrecemos permisos sobre la vista al usuario y no le damos permisos de ningún tipo sobre la tabla, que el usuario trabaje sobre las vistas.

Con esta actualización si me permitiría realizar modificaciones aunque tuviese with Check Option en la vista, ya que estoy trabajando sobre la Tabla.

```
Update emp set oficio = 'Nino' where oficio = 'Ventas'
```

Con esta actualización no podría realizar modificaciones sobre la vista, porque le he puesto With Check Option. De esta forma le doy permisos sobre la vista, pero no sobre la tabla, y no podrá modificar un cambio en el campo que yo he puesto.

```
Update Vendedores set oficio = 'Nino' where oficio = 'Ventas'
```

MODULO 9: IMPLEMENTACIÓN DE PROCEDIMIENTOS ALMACENADOS

- **With Recompile**

La primera vez que se ejecuta un procedimiento almacenado, se guardará en el cache del servidor, el plan de consulta a realizar (antes de realizar una consulta se elabora un plan de consulta para saber si realiza la consulta mediante índices o no y de que manera). Usando With Recompile, indicamos en el procedimiento almacenado que use el procedimiento de sistema **sp_recompile** para forzar la compilación del procedimiento cada vez que se ejecute con lo que se establecerá siempre un nuevo plan de consulta. Esto es muy práctico si por ejemplo una vez elaborado un plan de búsqueda de un procedimiento se añade un índice sobre una tabla que usa este, y ese índice puede ser de gran utilidad para elaborar el nuevo plan de búsqueda.

Sintaxis:
Create Procedure Nombre **With Recomplie** As Instrucciones SQL

- **SP_AddMessage**

  Añade mensajes de error al sistema. Lo añade en la tabla **sysmessages**. Sintaxis:
  SP_AddMessage Código, Gravedad, 'Mensaje', @Lang = 'Idioma'
  - *Código:* Todos los mensajes de error, llevan un código, este código hasta 50.000 está reservado para el sistema. A partir de 50.000 podremos insertarlo nosotros.
  - *Gravedad:* Código de gravedad del error.
  - *Idioma:* El idioma del mensaje.

  Nota Importante: Un mensaje puede tener el mismo código y el propio mensaje estar en diferentes idiomas. Todos los mensaje que creemos, deben estar al menos en inglés. Aunque si queremos, podemos insertar el mensaje con @Lang en inglés pero el mensaje se muestre en Español.

  Ejemplo:

```
SP_AddMessage 50020, 16, 'Hello', @Lang = 'US_English'
SP_AddMessage 50020, 16, 'Hola', @Lang = 'Spanish'
```

- **@@Error**
  Esta variable de sistema, coge el valor del error provocado por la última sentencia ejecutada, sino tiene error se introduce valor cero, si provoca error introduciría el código del error.

- **RaiseError**
  Lanza el mensaje indicado. Sintaxis:

```
RaiseError(Código, Severidad,Línea)
Código: El código del mensaje que queremos lanzar.
```

  Severidad: Nº que hace referencia a quién provoca el mensaje. A la hora de usar código de mensaje, introduciremos intervalos entre 11 y 16 que son los provocados por el usuario y que se pueden solucionar.
  Estado: Estado del mensaje.

Ejemplo:

```
SP_AddMessage 50020, 16, 'Hola', @Lang = 'Spanish'
Raiserror(50020,16,20)

Servidor: mensaje 50020, nivel 16, estado 20, línea 2
Hola
```

- **SP_dropMessage:** Elimina el mensaje de la base de datos

SP_dropMessage (Numero Mensaje, Idioma)

```
SP_dropMessage 50020,'Spanish'
```

- **Ver todos los mensajes**

Para ver los mensajes creados, debemos utilizar la tabla sysmessages de la base de datos Master:

```
use master
go
select * from sysmessages
```

- **Transacciones**

Para realizar una transacción seguimos la siguiente sintaxis:

```
Begin Transaction
Instrucciones
@@Error
RollBack Tran
Instrucciones
Commit Transaction
```

- *Begin Transaction:* Indica el comienzo de la transacción
- *@@Error:* Ya que controla si la última sentencia ejecutada ha producido error, lo pondremos detrás de cada consulta de acción para deshacer cambios si hay un error.
- @RollBack Tran: Deshace los cambios realizados mediante las consultas de acción.
- @Commit Transaction: Guarda los cambios realizados mediante las consultas de acción. Se introduce al final tal y como indica la sintaxis y en realidad siempre se realiza un commit, esto no significa que las consultas de acción realizadas antes del commit, se escriban, dependerá si se ha producido un error, ya que si el error está controlado, en cuanto se haga un rollback, es como si la consulta de acción no se hubiese ejecutado.

El siguiente ejemplo, muestra como manejar los errores y las transacciones:

Creamos una nueva base de datos llamada Almacen, y en ella creamos las siguientes tablas:

Introducimos los siguientes valores:

IDPRODUCTO	NOMBRE	UNIDADES
1	VEHÍCULO1	20
2	VEHÍCULO2	30

IDVENTA	IDPRODUCTO	UNIDADES
1	1	10

Añadimos los siguientes mensajes:

50010-> no se pudo insertar la nueva venta
50011 -> no se pudieron actualizar las unidades del almacen

```
sp_addmessage 50010, 16, 'NO SE PUDO INSERTAR LA NUEVA VENTA',
@Lang='US_English'
sp_addmessage 50011, 10, 'NO SE PUDIERON ACTUALIZAR LAS UNIDADES
DEL ALMACÉN', @Lang='US_English'
```

Creamos una regla check para que las unidades de almacen no puedan ser menor que cero.

```
Alter Table Almacen
Add Constraint Ck_Unidades
Check(Unidades > 0)
```

Creamos un procedimiento que introduciendole el nº de venta, el producto y las unidades realice la venta, y después reste las unidades vendidas del producto vendido de la tabla almacen..

```
CREATE Procedure NVentas
@IdVenta as int,
@IdProducto as int,
@Unidades as int
As
Begin Transaction
Insert Into Ventas Values(@IdVenta,@IdProducto,@Unidades)
if @@Error <> 0
        Begin
        RollBack Tran
        RaisError(50010,10,1)
        Return
End
Update Almacen set unidades = unidades - @unidades
where idproducto = @idproducto
If @@Error <> 0
        Begin
        Rollback Tran
        RaisError (50011,10,1)
        Return
End
Commit Transaction
```

Provocaremos los errores creados, insertando mas unidades de las que se pueden vender

```
Exec NVentas '1','1','50'
```

MODULO 10: FUNCIONES DEFINIDAS POR EL USUARIO

- **Declare Table**

Mediante esta sentencia, se pueden crear tablas temporales en memoria, y posteriormente manipularlas como si fueran tablas normales. Sintaxis:
Declare @Tabla Table
(Campo1 tipo, Campo2 tipo, Campo3 tipo...)
Estas tablas temporales solamente sirven para el momento de la ejecución, si se intentan utilizar en otro momento el sistema no las reconocerá.

El siguiente ejemplo crea la tabla temporal y después inserta en ella:

```
Declare @Emp Table
(Emp_no int Primary Key,
Apellidos nvarchar(25),
Salario Int)
Insert Into @Emp
Select Emp_No, Apellido,
Salario from Emp
Select * from @Emp
```

	Emp_no	Apellidos	Salario
1	7499	ARROYO	1067630
2	7521	SALA	834285
3	7654	MARTIN	1048316
4	7698	NEGRO	1921187
5	7782	CEREZO	1684299
6	7839	REY	2337500
7	7844	TOVAR	1001030
8	7900	JIMENO	633985

- **FUNCIONES DE USUARIO**

Una función de usuario, se crea con el fin de automatizar una consulta que se realiza a menudo. Pueden usar uno o más parámetros de entrada y devuelven un valor o varios resultados. Existen dos tipos de funciones:

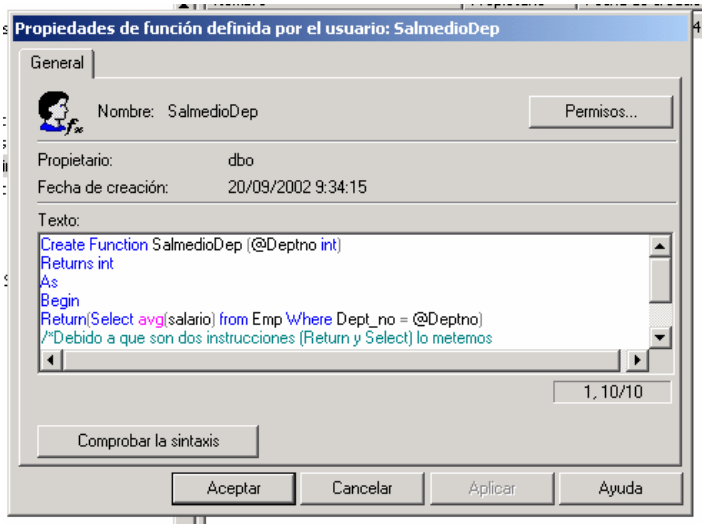
- **Escalares**

Devuelven un solo valor. Sintaxis:

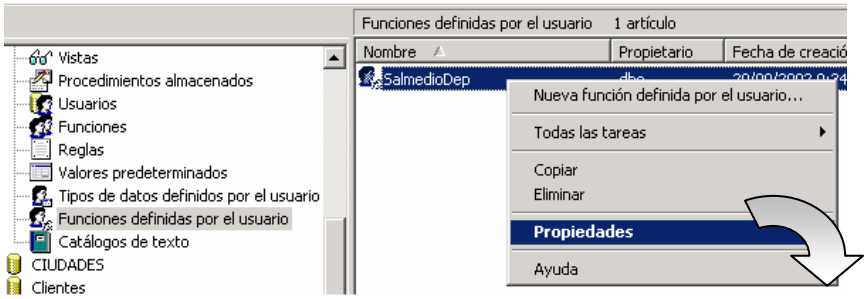
```
Create Function NombreFunción (@Parámetro1 tipo, @Parámetro 2 tipo...)
Returns Tipo
As
Begin
Sentencias
End
```

Ejemplo:

```
Create Function SalmedioDep (@Deptno int)
Returns int
As
Begin
Return(Select avg(salario) from Emp Where Dept_no = @Deptno)
End
```



Esta función devuelve el salario medio del departamento indicado.



Para ver si la función se ha creado, consultamos el apartado “Funciones definidas por el usuario” del administrador corporativo. Si la función no aparece, pulsamos F5 para que refresque.

```
select * from Emp
where salario > dbo.salmediodep(30) and dept_no <> 30
```

	EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
1	7782	CEREZO	DIRECTOR	7839	1978-06-09 00:00:00	1684299	0	10
2	7839	REY	PRESIDENTE	0	1975-11-17 00:00:00	2337500	0	10

Como vemos en el ejemplo, **cuando ejecutemos la función hemos de poner también el usuario al que pertenece**, en este caso es el dbo.

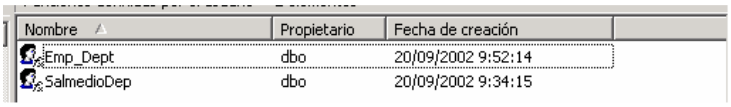
- **Tabla**

Estas funciones devuelven una tabla. Sintaxis:
Create Function NombreFunción(Parámetro1 tipo, Parámetro2 tipo...)
Returns Table
As
Return(Sentencias)
/
Sentencias
Return

Como vemos, en la sintaxis, esta sentencia puede construirse de diferentes maneras. Return indica que tiene que devolver un valor. Usaremos una sintaxis u otra en los siguientes casos:
Si tenemos que hacer una sola sentencia o una sentencia anidada, usaremos Return (Sentencia / Sentencias anidadas) en cambio si dependiendo de una serie de condiciones controladas mediante if, when, etc. realizamos unas sentencias u otras, incluiremos Return al final del todo.

Ejemplo 1:

```
Create Function Emp_Dept (@Ndept nvarchar(20))
Returns Table
As
Return(Select Emp_No, Apellido, Salario, DNombre From Emp
Inner Join Dept
On Dept.Dept_no = Emp.Dept_no
where DNombre = @NDept)
```



Nombre	Propietario	Fecha de creación
Emp_Dept	dbo	20/09/2002 9:52:14
SalmedioDep	dbo	20/09/2002 9:34:15

Si vamos a Funciones de usuarios veremos la función que hemos creado.

Después tratamos la función como si fuera una tabla, ej:

```
Select * from Emp_Dept ('Ventas')
```

En este caso introducimos como valor Ventas, y vemos como obtenemos todos los empleados del departamento indicado.

	Emp_No	Apellido	Salario	DNombre
1	7499	ARROYO	1067630	VENTAS
2	7521	SALA	834285	VENTAS
3	7654	MARTIN	1048316	VENTAS
4	7698	NEGRO	1921187	VENTAS
5	7844	TOVAR	1001030	VENTAS
6	7900	JIMENO	633985	VENTAS

Ejemplo 2:

```
Create Function Emp_Dept2(@ndept nvarchar(20))
Returns @Emp Table(Emp_no int, Apellido nvarchar(20), Salario int,
Dnombre nvarchar(20))
As
Begin
  Insert Into @Emp
  Select Emp_no, Apellido, Salario, Dnombre from Emp
  Inner Join Dept
  On Dept.Dept_no = Emp.Dept_no
  where Dnombre = @NDept
Return
End

Select * from Emp_dept2 ('Ventas')
```

	Emp_no	Apellido	Salario	Dnombre
1	7499	ARROYO	1067630	VENTAS
2	7521	SALA	834285	VENTAS
3	7654	MARTIN	1048316	VENTAS
4	7698	NEGRO	1921187	VENTAS
5	7844	TOVAR	1001030	VENTAS
6	7900	JIMENO	633985	VENTAS

- **Drop Function**

Borra una o varias funciones. Sintaxis
Drop Function Funcion1, Funcion2 etc.

Ejemplo:

```
Drop Function SalMedioDep, Emp_Dept, Emp_Dept2
```

- **Create Function With SchemaBinding**

Crea funciones con dependencias, para poder proteger la estructura de las tablas usadas en una función. Es decir usando with schemabinding, no se podrá usar Alter ni Drop en las tablas de las que depende.

```
Create Function Emp_Dept (@Ndept nvarchar(20))
Returns Table
With SchemaBinding
As
Return(Select Emp_No, Apellido, Salario, DNombre From Dbo.Emp
Inner Join Dbo.Dept
On Dept.Dept_no = Emp.Dept_no
where DNombre = @NDept)
```

Como vemos en el ejemplo al crear las tablas con dependencias, hemos de indicar cuál es su propietario.

Si intentamos borrar o modificar la tabla nos mostrará un error.

```
Drop Table Emp
```

```
Servidor: mensaje 3729, nivel 16, estado 1, línea 1
No se puede DROP TABLE 'Emp' porque el objeto 'Emp_Dept' le hace referencia.
```

- Funciones No deterministas: Trabajan siempre con el mismo tipo de valor pero devuelven cada vez un valor diferente. Ej. getdate() trabaja siempre con la fecha del sistema pero devuelve un valor diferente, @@Error, siempre trabaja con los errores del sistema pero devuelve errores diferentes.
  A la hora de crear funciones definidas por el usuario, no podemos usar funciones no deterministas.

  Ejercicios:

  1. Crear una función que pasándole una fecha y un separador a elegir nos debe cambiar el separador de la fecha que le estamos pasando por el separador elegido, ejemplo pasándole la fecha 01/12/2001 y el separador ":" obtendremos 01:12:2001. hemos de tener en cuenta que la fecha que le pasemos no tiene porque tener el separador /.
     Para probar que la función es correcta, seleccionaremos todas las fechas de alta de la tabla Empleados y le pasaremos el separador ":"

```
Create Function CambiaFecha(@fecha datetime, @separador nvarchar(1))
Returns nvarchar(10)
Begin
Return
(Convert(nvarchar(6),(Day(@fecha)))
+ @separador
+ Convert(nvarchar(6),(Month(@fecha)))
+ @separador
+ Convert(nvarchar(6),(Year(@fecha))))
End
```

Al declarar la función usamos convert para convertir el día, mes o año a caracter, ya que si intentamos concatenar números con carácteres da error porque piensa que intentamos sumar.

```
Select Dbo.CambiaFecha(fecha_alt, '*') as Resultado from Emp
```

```
Resultado
----------
17*12*1980
11*12*1976
22*2*1981
22*2*1981
2*4*1981
```

2. Crear una función que pasándole el parámetro ‘Completo’ Seleccione el apellido junto su número de empleado en una columna y en otra el departamento y pasándole ‘Apellido’ Seleccione el apellido y el departamento.

```
create function Empleados(@Param nvarchar(50))
Returns @Tabla table
(Empleado nvarchar(50)
,Dept_no int)
as
Begin
        if (@Param = 'Completo')
                insert into @Tabla
                Select Apellido + ' ' + Convert(char(6),emp_no)
                , dept_no from emp
        else
                insert into @Tabla
                Select Apellido, Dept_no
                from emp
        Return
end
```

```
Select * from dbo.Empleados('Completo')
```

	Empleado	Dept_no
1	SANCHEZ 7369	20
2	ARROYO 7499	30
3	SALA 7521	30
4	JIMENEZ 7566	20
5	MARTIN 7654	30
6	NEGRO 7698	30
7	CEREZO 7782	10
8	GIL 7788	20
9	REY 7839	10
10	TOVAR 7844	30
11	COALESCE 7861	10

```
Select * from dbo.Empleados('Otro')
```

	Empleado	Dept_no
1	SANCHEZ	20
2	ARROYO	30
3	SALA	30
4	JIMENEZ	20
5	MARTIN	30
6	NEGRO	30
7	CEREZO	10
8	GIL	20
9	REY	10
10	TOVAR	30
11	COALESCE	10

3. Crear una función que dependiendo de los datos que le enviemos, nos devolverá un informe sobre los empleados. Los parámetros que le podemos enviar a la función son: Nº Departamento, Nº Empleado, Fecha u Oficio. Dependiendo del dato, mostraremos unos datos u otros.

El señor Sanchez con cargo de Empleado se dió de alta el 17 de Diciembre de 1980.

```
create function Empleados(@Param nvarchar(50))
Returns @Tabla table
(Empleado nvarchar(200))
as
Begin
    if (isnumeric(@Param) = 1)
            insert into @Tabla
            select isnull('El señor ' + LTRIM(cast(apellido as nvarchar(15)))
            + ' con cargo de '
            + ltrim(cast(oficio as nvarchar(15)))
            + ' se dió de alta el '
            + cast(day(fecha_alt) as char(2)) + ' de '
            + ltrim(cast(datename(month,fecha_alt) as nvarchar(15)))
            + ' de '
            + cast(year(fecha_alt) as char(4))
            ,'EMPLEADO SIN NOMBRE')
             as [DATOS EMPLEADOS]
            from emp
            where emp_no = @Param
    else
    begin
            if (isdate(@Param) = 1)
            BEGIN
            insert into @Tabla
            select isnull('El señor ' + LTRIM(cast(apellido as nvarchar(15)))
            + ' con cargo de '
            + ltrim(cast(oficio as nvarchar(15)))
            + ' se dió de alta el '
            + cast(day(fecha_alt) as char(2)) + ' de '
            + ltrim(cast(datename(month,fecha_alt) as nvarchar(15)))
            + ' de '
            + cast(year(fecha_alt) as char(4))
            ,'EMPLEADO SIN NOMBRE')
             as [DATOS EMPLEADOS]
            from emp
            where fecha_alt > @Param
            END
            ELSE
            BEGIN
            insert into @Tabla
            select isnull('El señor ' + LTRIM(cast(apellido as nvarchar(15)))
            + ' con cargo de '
```

```
            + ltrim(cast(oficio as nvarchar(15)))
            + ' se dió de alta el '
            + cast(day(fecha_alt) as char(2)) + ' de '
            + ltrim(cast(datename(month,fecha_alt) as nvarchar(15)))
            + ' de '
            + cast(year(fecha_alt) as char(4))
            ,'EMPLEADO SIN NOMBRE')
             as [DATOS EMPLEADOS]
            from emp
            where apellido like @Param
            END
    end
    Return
end
```

MODULO 11: IMPLEMENTACION DE DESENCADENADORES

- **Trigger Instead Of**

Es un tipo de trigger, que asociado a una vista, cuando se intenta realizar el tipo de consulta que indica el trigger (insertar, modificar, o eliminar), una vez están los registros en las tablas inserted o deleted, la consulta se interrumpe y salta el trigger, con lo que podemos manejar los datos que hay en estas tablas temporales mediante el trigger, esto es muy práctico cuando queremos insertar en varias tablas pertenecientes a una vista, ya que con una simple consulta no podremos, tendremos que trabajar con un trigger Instead Of y usar las filas incluidas en la tabla inserted.

Sintaxis:

Create Trigger NombreTrigger
On Vista
Instead Of
As
Sentencias SQL

Un ejemplo práctico sería el siguiente:

Tenemos una vista que hace referencia a Departamentos y empleados, y contiene el nº del departamento y nombre de la tabla departamentos y el nombre y apellidos de la tabla empleados. Si intentamos hacer un insert sobre la vista, sin el trigger dará error y no dejará insertarlo, pero en cambio, con el triger:
Cuando hagamos la inserción salta el trigger. Mediante el trigger, accedemos a la tabla inserted, e insertamos los valores necesarios en la tabla departamentos y empleados.

Ejemplo:

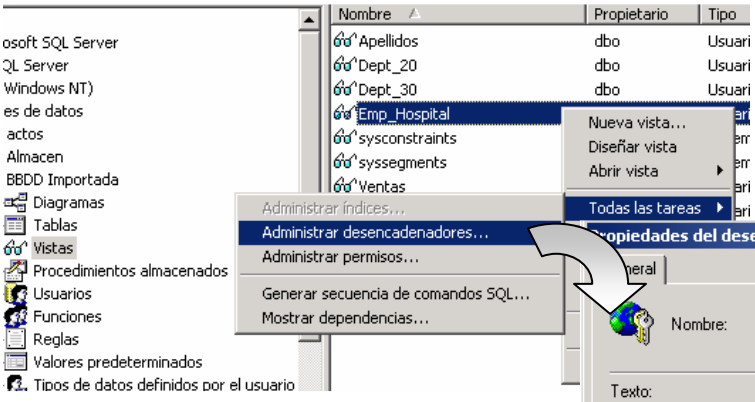
Creamos una vista que combine la tabla Doctor y Plantilla

```
CREATE View Emp_Hospital
As
Select Doctor_no, Apellido, Especialidad, Hospital_Cod
From Doctor
Union
Select Empleado_no, Apellido, Funcion, Hospital_Cod
From Plantilla
```

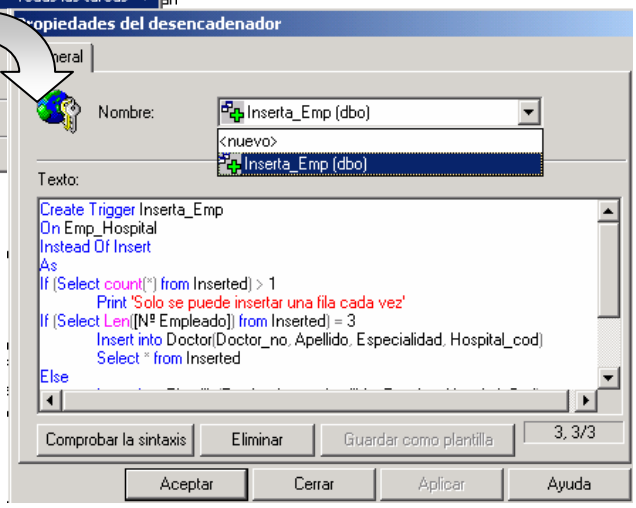
Creamos un trigger para esta vista.

```
Create Trigger Inserta_Emp
On Emp_Hospital
Instead Of Insert
As
If (Select count(*) from Inserted) > 1
        Print 'Solo se puede insertar una fila cada vez'
If (Select Len([Doctor_no]) from Inserted) = 3
        Insert into Doctor(Doctor_no, Apellido, Especialidad, Hospital_cod)
        Select * from Inserted
Else
        Insert into Plantilla(Empleado_no, Apellido, Funcion, Hospital_Cod)
        Select * from Inserted
Return
```

Para ver un trigger de una vista mediante el administrador Corporativo:



Dentro de vistas, seleccionamos la deseada, y elegimos la opción Todas las taréas -> Administrar desencadenadores...



...posteriormente, ya en la pantalla de desencadenadores, en nombre elegimos el desencadenador que queramos.

Ejemplo2:

Nº EMPLEADO, APELLIDO, OFICIO Y NOMBRE DE DEPARTAMENTO

Al insertar sobre la vista el siguiente registro:
insert into VistaEmpDept (Emp_no, Apellido, Oficio, Dnombre) Values ('8888','Angulo','Vendedor','Investigación')

El trigger instead of saltará e insertará el empleado con su correspondiente nº de departamento, ya que en empleados no hay nombre de departamento, con lo que averiguaremos el nº de departamento haciendo una select sobre la tabla de departamentos.

```
Create Trigger Inserta_Emp
On TriggerEmpDept
Instead Of Insert
As
declare @Dept int
select @dept = dept_no from dept,inserted
where dept.dnombre = inserted.dnombre
if @dept is null
  Begin
        print 'No existe departamento con ese nombre'
        print 'No se ha realizado inserción'
  End
else
        insert into emp (emp_no,apellido,fecha_alt,dept_no)
        select inserted.emp_no,inserted.apellido
        ,inserted.fecha_alt,@dept from inserted
```

Si realizo ahora la inserción sobre la vista, el empleado se va a insertar primero sobre la tabla temporal, yo evaluaré los datos que se han insertado en la tabla temporal para decidir si me sirven o no. En este caso en concreto no me sirven porque estoy insertando un departamento que no existe, Investigación está sin acento en la tabla.

```
insert into TriggerEmpDept (Emp_no, Apellido, fecha_alt, Dnombre)
Values (8888,'Angulo','12/12/99','Investigación')
```

```
No existe departamento con ese nombre
No se ha realizado inserción

(1 filas afectadas)
```

Si cumplo los requisitos que yo he puesto en el trigger sobre la vista, me permitirá insertar el empleado.

```
insert into TriggerEmpDept (Emp_no, Apellido, fecha_alt, Dnombre)
Values (8888,'Angulo','12/12/99','Investigacion')
```

	EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
1	8888	Angulo	NULL	NULL	1999-12-12 00:00:00	NULL	NULL	20

```
Create Trigger Inserta_Empleado
On Emp
Instead Of Insert
As
declare @dir int
declare @emp_no int
declare @oficio nvarchar(30)
select @dir = inserted.dir from inserted
select @emp_no = emp_no, @oficio = oficio from emp
where emp_no = @dir
if @emp_no is null
begin
        print 'Fallo, no se ha realizado la inserción'
        print 'El empleado insertado no tiene director adecuado'
end
else
begin
        if @oficio = 'DIRECTOR' OR @OFICIO = 'PRESIDENTE'
        BEGIN
                insert into emp
                select * from inserted
        END
        ELSE
        BEGIN
                print 'Fallo, no se ha realizado la inserción'
                print 'El dir no es director'
        END
end
Return
```

INDICES

Es una estructura auxiliar que sirve para optimizar las consultas. Mediante esta estructura, al realizar una consulta SQL Server realizará menos operaciones para devolver los resultados y la consulta se realizará mas rápidamente ya que los datos están estructurados de forma que sea mas sencillo localizarlos.
Algo muy similar a los Indices creados por Sql Server, es el índice de un libro, en una página indica en que página del libro está cada capítulo, y además podemos saber también, que apartados hay dentro de un capítulo y en que página están. En el caso de una base de datos, todos los registros de esta, se almacenan en páginas, después se crean unos índices que indican en que página de datos está cada registro y por encima de estos índices, se crean otros subíndices que almacenan la cabecera de cada índice. Además se crearía una cabecera principal que almacenaría a su vez la cabecera de estos subíndices.

Como se almacenan los datos

- Cada registro de cada tabla, se almacena en una página de datos estas páginas tienen un tamaño de 8 kb. A un grupo de ocho páginas de datos se le llama **Extensión**.
- Las filas no se almacenan en un orden concreto y no hay un orden detallado de la secuencia de las páginas de datos.
- No existe una lista que almacene la estructura de las páginas de datos.
- Cada vez que se inserta un nuevo registro en una página de datos llena, esta se fracciona creandose otra página de datos.

Como se accede a los datos

1. SQL Server verifica si existen índices
2. En caso de que si haya índices, el optimizador de consultas verifica si es mas sencillo realizar la consulta usando los índices o sin usarlos

Sin indices:
- Va a la primera hoja de datos de la tabla
- Busca página por página examinando todas las filas de cada página
- Selecciona las filas que cumplen la consulta

Con índices:
- Usa la estructura de árbol del índice para buscar las filas que cumplen la consulta.
- Una vez sabe donde están extrae estas filas.

Ventajas y desventajas de los índices

Los indices abrevian la recuperación de los datos, sin índices tendríamos que recorrer todas las páginas de datos hasta encontrar las filas que cumplen la consulta, sobre todo aceleran las consultas que combinen varias tablas y agrupen u ordenen datos.

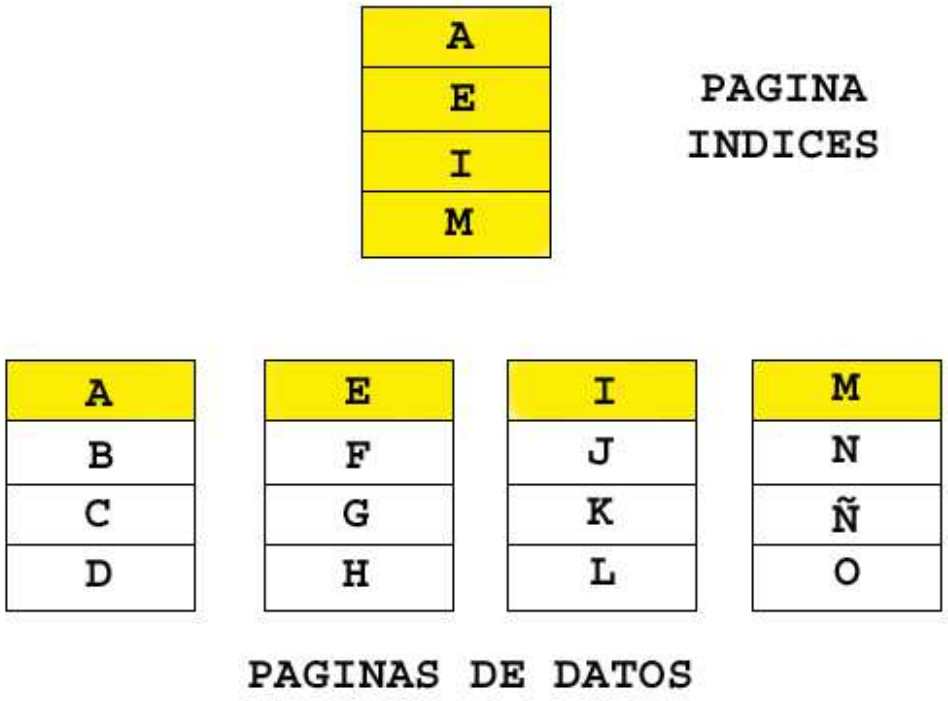
Como almacena los datos la base de datos:

- **Si los datos estan sin ordenar se llaman montón. Un montón va a ser un grupo de páginas y cada página tiene 8Kb.**
- **Si no tengo indices en la tabla recorro toda la tabla, con lo cual tengo que realizar muchas idas y salidas de datos en las páginas.**
- **Con un indice busca los datos con un orden lógico interno, sin necesidad de buscar en todas las páginas.**
- **Una página solo puede tener datos de una tabla.**
- **Cuando creamos un indice incrementamos el tamaño de la base de datos.**

Existen dos tipos de indices:

- Agrupados:
  - Un indice agrupado por tabla.
  - Los datos se reordenan por el campo indexado.
  - Los datos se colocan en orden alfabético hasta ocupar los 8 Kb que ocupa cada página.
  - Cuando una página está completa, se divide en otra página para dar entrada a nuevos datos.
  - Cuando inserto un indice nuevo, las páginas antiguas se borran y se crean las nuevas páginas ya ordenadas alfabéticamente.
  - La estructura se divide en varios niveles, por un lado está el nivel de hoja de los datos, otro nivel lo forman las páginas de indice y el último nivel está formado por la raiz.
  - Una página solo puede tener un nivel raiz, pero puede tener más de un nivel de hoja de indices.
  - La página de indices y la página de datos coinciden en su orden de colocación.
  - En el momento de insertar datos, se dividen las páginas para poder insertar el dato colocado alfabéticamente, el problema es que se fragmenta la página de datos y la página de indices, esto provoca que se tengan que hacer búsquedas más largas, debido a que buscamos en el indice, en la página de datos y en las páginas fragmentadas.
  - Los indices agrupados no deben realizarse sobre columnas que tengan muchas modificaciones y actualizaciones, debido a que los datos se fragmentaran.
  - Realizar indices agrupados sobre columnas en las que se realicen búsquedas exactas o intervalos de valores.
  - Un indice agrupado sería como un índice de un Diccionario, busco por la letra ordenada alfabeticamente y encuentro la palabra que deseo en muy poco tiempo.
  - La aplicación no siempre utiliza el indice, depende de la consulta y no podemos controlar que utilice el indice. Normalmente siempre lo utiliza.
  - Se puede realizar sobre columnas con datos duplicados, aunque conviene que sea sobre datos que no esten muy repetidos.
  - En caso que hubiese elementos duplicados, se genera un indice único interno propio del indice.
  - Se puede dejar un espacio en las páginas para que no las fragmente al insertar nuevos datos.

GRAFICO DE INDICES AGRUPADOS

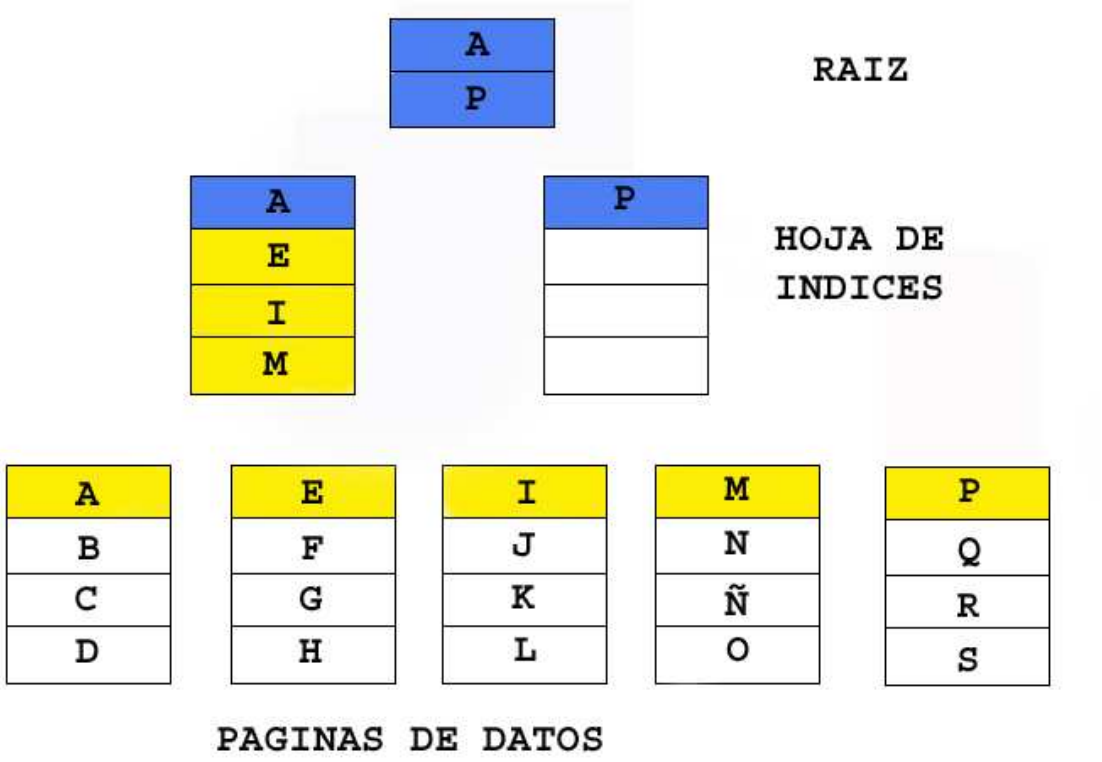


¿Cuantas páginas de indices se pueden generar?
Las páginas de indices tambien son de 8Kb, con lo cual, cuando se llena un indice, se crea otra pagina de indices a su lado.
Los 8Kb de cada página son de tamaño fisico, no en memoria.

Cuando hacemos una Primary Key, generamos un indice agrupado, aunque también se puede decir que no lo haga poniendo después de Primary Key la palabra **nonclustered**.

Ejemplo de indices agrupados:

GRAFICO DE UN INDICE AGRUPADO COMPLETO



Realizo una consulta por el campo indexado: Busco **H**.

La búsqueda comienza por la raiz, compara el valor a buscar **H** con el dato A, selecciona si el dato que busca es mayor o menor que el que compara. Como el dato **A** es menor, continua con el dato **P**. Como el dato **P** es mayor al dato **H**, vuelve y ya sabe en que página del indice debe buscar el dato. En la página de indices repite el mismo proceso, compara con el dato **A**, compara con el dato **E** y compara con el dato **I**, como el dato **I** es mayor al dato **H**, vuelve y sabe que el dato que encuentra está en la página del dato **E**. Busca en la página **E** hasta encontrar el dato.
Con este sistema solamente recorre dos páginas, teniendo las páginas sin indices recorrería todos los datos hasta dar con la **H**, devolviendo los datos de la consulta.

Ejemplo para calcular de forma teorica el tamaño que puede ocupar un indice en una tabla:

Tenemos una Tabla con 10000 PAGINAS Y 50000 FILAS

Queremos hacer un indice a partir de una tabla en la que cada tipo de dato en la columna de tiene un tamaño fijo de 5 bytes. Por ejemplo varchar(5).
CREAR CLAVE AGRUPADA 5 BYTES LONGITUD FIJA

La Hoja del indice ocupará 12 bytes porque el índice incluye información adicional, una clave interna y unica para cada dato que tiene el indice.
Dependiendo del dato, añade de 5 a 8 bytes más en cada página de indice.
Ahora sabemos cuantas filas ocupan y el tamaño de cada página.

Cada página de indice tiene un tamaño fijo de 8096 bytes.
Cada fila del indice ocupa 12 bytes.
Haciendo este cálculo, sabemos cuantas filas de datos va a contener cada página de indice.

8096 bytes / 12 bytes = 674 filas en cada indice, en el momento que pongamos 675 datos, la página se dividirá.

Cada una de las páginas va a generar una fila en las páginas de indice, habrá que dividir las 10000 páginas por las 674 filas y sabremos cuantas páginas de indices hay.

10000 páginas / 674 Filas = 15 páginas

Nivel de hoja de datos:
10000 páginas

Nivel del indice: 15 Paginas

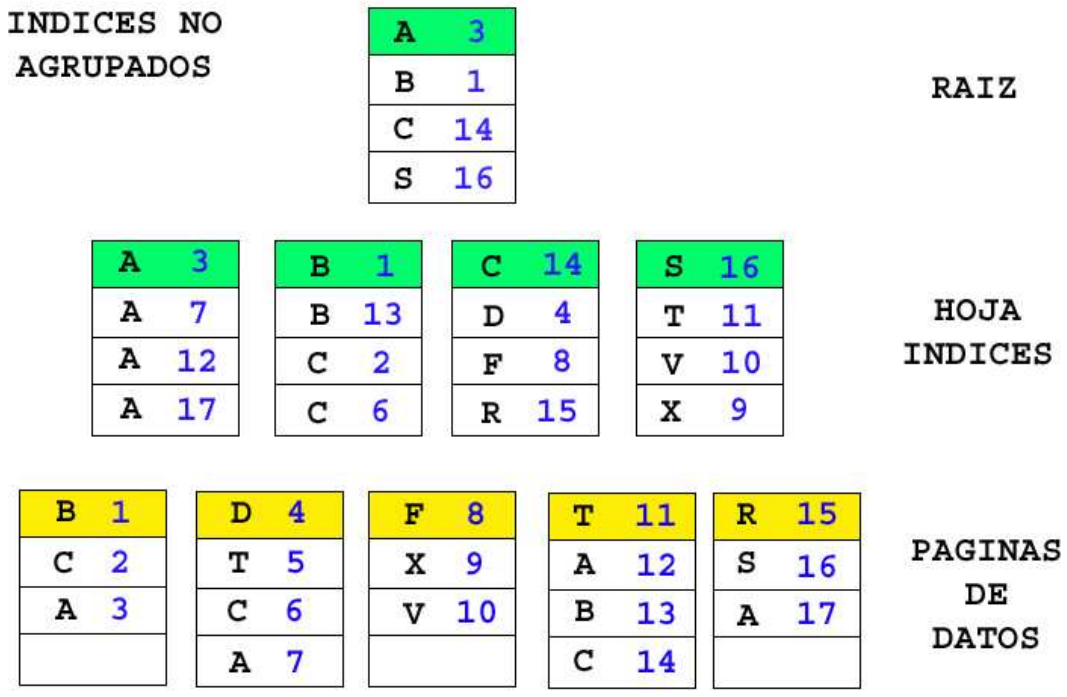
Se va a generar otro nivel que es nivel 2 que es raiz, porque el nivel raiz solamente va a tener 15 filas, pero tiene espacio para 674.

Nuestro archivo de datos se configura asi, si tenemos 10000 páginas y dividimos por 16 páginas de indice, sale un 1 por ciento que ocupa de espacio el indice, a partir de 3% se supera el limite de coste de un indice, entonces hay que plantearse si se deben utilizar indices en la tabla o no.

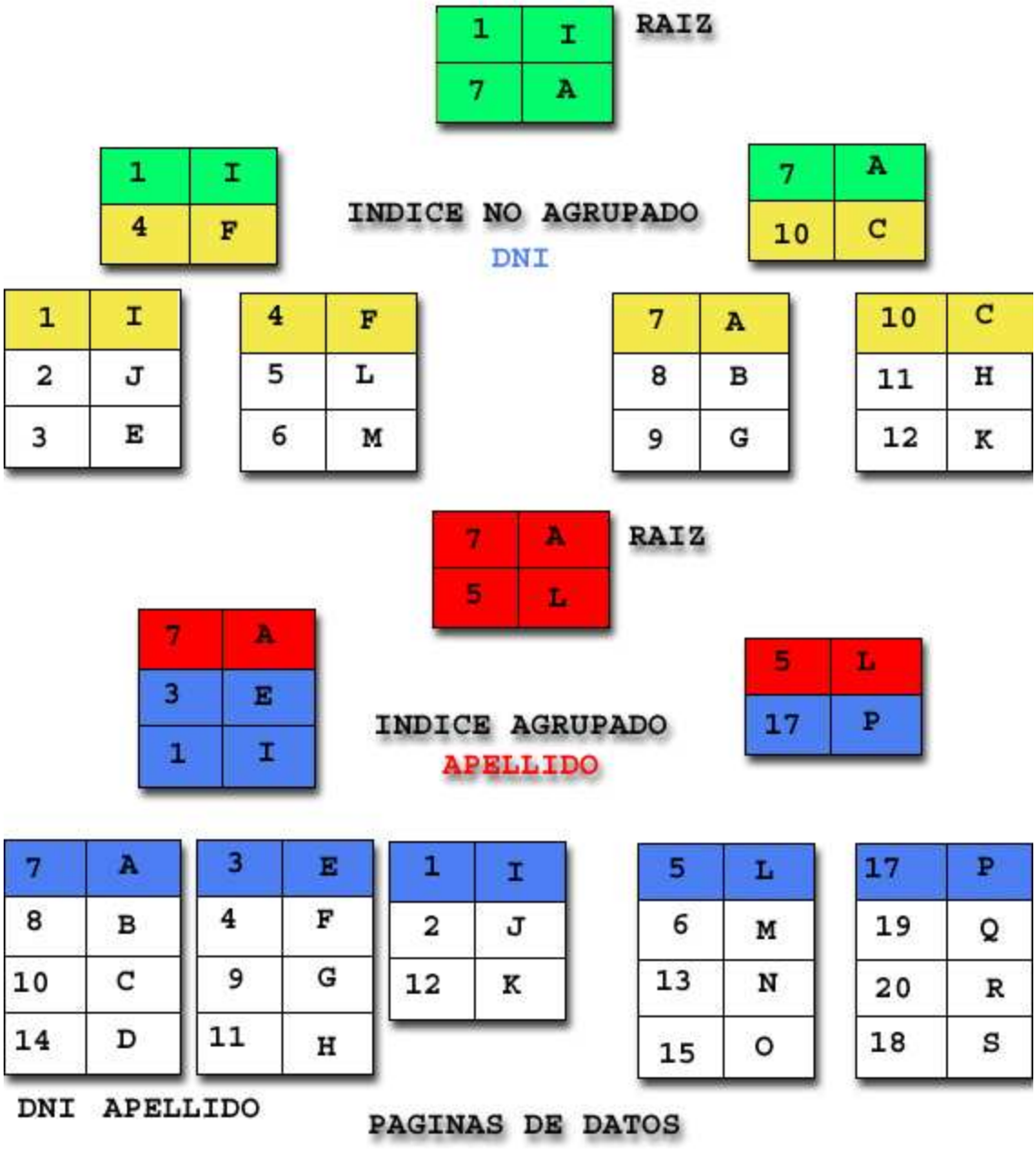
- No Agrupados:

  - Si se insertan filas después de generarse los índices, se inserta al final y no están incluidos en los índices.
  - Almacena todas las filas desordenadas, con un orden en el indice.
  - Posteriormente crea los índices que contienen un puntero (identificador unico) que indica en que página de datos y que fila está cada registro. Por encima de estos hay otros índices que contienen la cabecera de cada uno de ellos y por encima queda un único índice llamado raiz que contiene la cabecera de cada indice inferior. La búsqueda se realizará internamente por este identificador único.
  - Lo bueno que tiene este indice es que sabe cuantos saltos debe dar para ir a las consultas, sabe exactamente cuantos valores va a devolver.
  - Es mejor utilizar indices no agrupados en las tablas que sufran muchas modificaciones, inserciones y borrado de datos, debido a la fragmentación de los datos.
  - Los datos borrados se llaman datos fantasma, estos datos permanecen en las hojas de indice un tiempo, cada cierto periodo son eliminados, el ordenador lo hace automaticamente.
  - Los indices no agrupados corresponderían a una especie de glosario, donde buscamos unas palabras en un libro y nos muestra las páginas donde estan situadas dichas palabras.

GRAFICO DE INDICES NO AGRUPADOS



Combinación de una tabla con un indice agrupado y otro no agrupado.



- Sirven para buscar valores en otras columnas que no estén agrupadas por ningún tipo de indices.
- Se pueden crear tantos indices no agrupados como se deseen, pero solamente puede haber un único indice agrupado.

Select oficio from emp where dni = 7
Oficio es un campo de la tabla que no tiene indice, pero el campo DNI sí que tiene un índice no agrupado.
Para esta consulta, la búsqueda comenzaria por el indice no agrupado, iría buscando por el DNI hasta encontrar la ruta para llegar al índice agrupado, dónde se realizaría la búsqueda por el campo apellido para llegar más rápido al dato que buscamos. Una vez encontrado el dato en las páginas de datos, se devuelve el campo oficio.

Elegir el tipo de índices a usar

Todos los indices se almacenan en la tabla Sysindexes.

```
use master
go
select * from sysindexes
```

id	status	first	indid	root	minlen	keycnt	groupid	dpages	reserved	use
1	18	0x0800000000100	1	0x0C0000000100	42	1	1	23	46	41
1	2	0xC80500000100	2	0xCA0500000100	7	3	1	9	16	11
1	0	0xC30500000100	3	0xC60500000100	9	2	1	3	5	5
2	18	0x1800000000100	1	0x1A0000000100	82	2	1	4	6	6
2	0	0x4800000000100	255	0x4800000000100	0	0	1	0	16	14
3	16402	0x1000000000100	1	0x1100000000100	46	3	1	67	97	94

La columna id, identifica el indice en la tabla.
La columna INDID, establece el tipo de indice que es:

- **1 Agrupado**
- **2 – 255 No agrupado**
- **0 Es un valor que almacena para las Primary Key.**

Selectividad = Nº total de filas – nº de filas devueltas.
Si tenemos muchos datos duplicados hay una alta densidad
Si tenemos pocos datos duplicados hay baja densidad

MODULO 7: CREACIÓN Y MANTENIMIENTO DE ÍNDICES

- **Create Index**

Crea índices igual que con el administrador corporativo de SQL Server. Para ejecutar esta instrucción es imprescindible ser el propietario de la tabla. No se pueden crear índices de vistas.

Create Clustered / **NonClustered**
Index Nombre_Indice0
On Tabla(Columna) **Asc** / **Desc**

Para los indices agrupados se utilizará el Prefijo CL_ antes del nombre del indice.

```
Create Clustered
Index CL_Emp
On Emp(emp_no) asc
```

Para los indices no agrupados se usará el Prefijo NonCL_ antes del nombre del indice.

```
Create NonClustered
Index NonCL_Apellido
On Emp(Apellido)
```

FillFactor

Si no decimos nada, las hojas de indices se llenan completamente cada vez que añadimos sobre la base de datos.
Si una hoja de indice está llena, la hoja de indice se fragmenta.
Fillfactor es una instrucción que nos permite dejar un espacio porcentual en cada página para las posibles inserciones, de esta forma se evitan las fragmentaciones de página y de indices. Solamente se fragmentarán cuando hayamos rellenado el espacio que hayamos puesto con Fillfactor.
Si ponemos Fillfactor, dejamos un 70% de espacio libre en las hojas de indices.
Valor predeterminado es 0. (Si no se pone nada)
La fragmentación aumenta espacio y además aumentan los saltos entre hojas.

Sintaxis:

```
Create Clustered
Index CL_Emp
On Emp(emp_no)
With Fillfactor=70
```

Sp_helpindex Tabla

- Dice el nombre del indice, la descripción y la columna a la que está asociado.

```
Sp_helpindex emp
```

index_name	index_description	index_keys
CL_Emp	clustered located on PRIMARY	EMP_NO

DBCC SHOWCONTIG (Tabla, NombreIndice)

```
USE NORTHWIND
GO
DBCC SHOWCONTIG ('ORDERS', PK_ORDERS)
```

```
DBCC SHOWCONTIG recorriendo la tabla 'Orders'...
Tabla: 'Orders' (21575115); Id. de índice: 1, Id. de base de datos: 6
Realizado recorrido de nivel TABLE.
- Páginas recorridas.............................: 20
- Extensiones recorridas.........................: 5
- Cambios de extensión...........................: 4
- Promedio de páginas por extensión..............: 4.0
- Densidad de recorrido [Cuenta óptima:Cuenta real].....: 60.00% [3:5]
- Fragmentación del recorrido lógico ..................: 0.00%
- Fragmentación del recorrido de extensión ..................: 40.00%
- Promedio de bytes libres por página.....................: 146.5
- Promedio de densidad de página (completa).....................: 98.19%
Ejecución de DBCC completada. Si hay mensajes de error, consulte al administrador del
sistema.
```

- **Páginas recorridas:** Número de páginas de la tabla o el índice.
- **Extensiones Recorridas:** Número de extensiones de la tabla o el índice.
- **Cambios de extensión:** Número de veces que la instrucción DBCC se movió desde una extensión a otra mientras recorría las páginas de la tabla o del índice.
- **Densidad de recorrido:** Cuanto más cerca esté del 100% mejor será la estructura del indice.
- **Promedio de bytes libres por página.:** Valor promedio de los bytes libres de las páginas examinadas. Cuanto mayor sea este valor, menos páginas llenas hay. Los valores bajos son buenos indicadores.
- **Promedio de densidad de página**. Este valor tiene en cuenta el tamaño de la fila, de forma que es una medida más precisa del grado de ocupación de las páginas. Cuanto mayor sea el porcentaje, mejor.

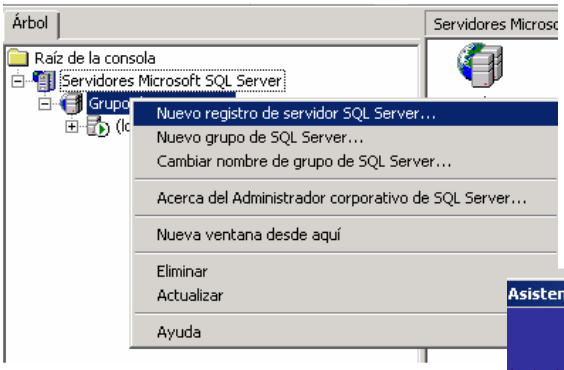
Las extensiones son las páginas que se han fragmentado después de crear el indice.

Drop Index

Borra índices
Drop Index Tabla.Nombre_Indice

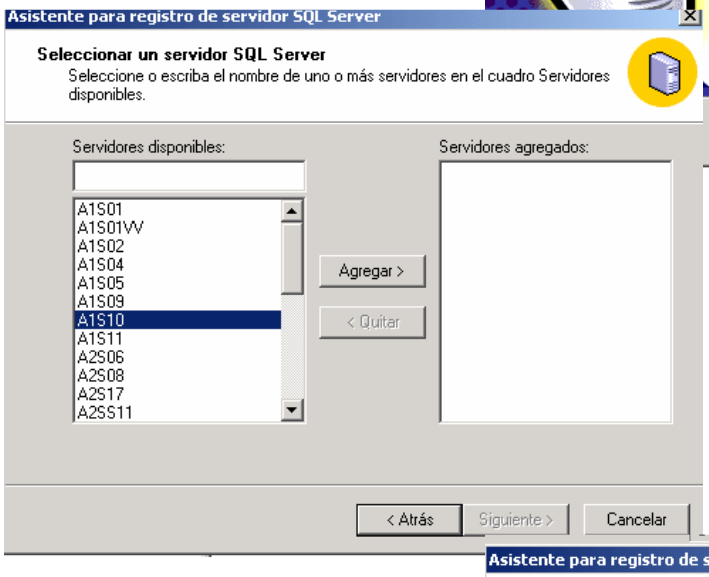
Drop index **E**mp.CL_Emp

Trabajar con varios Servidores



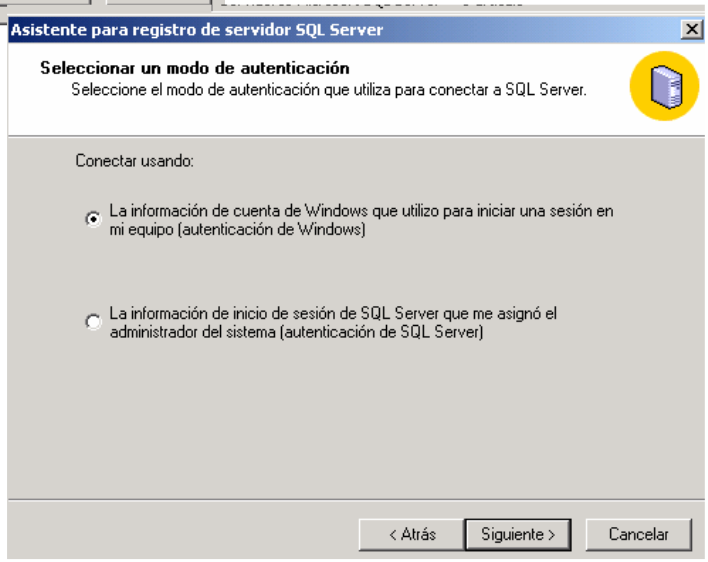
Dentro del Administrador de servidores, pulsamos botón derecho en grupo de Sql Server, y elegimos la opción "Nuevo registro de servidor SQL Server..."

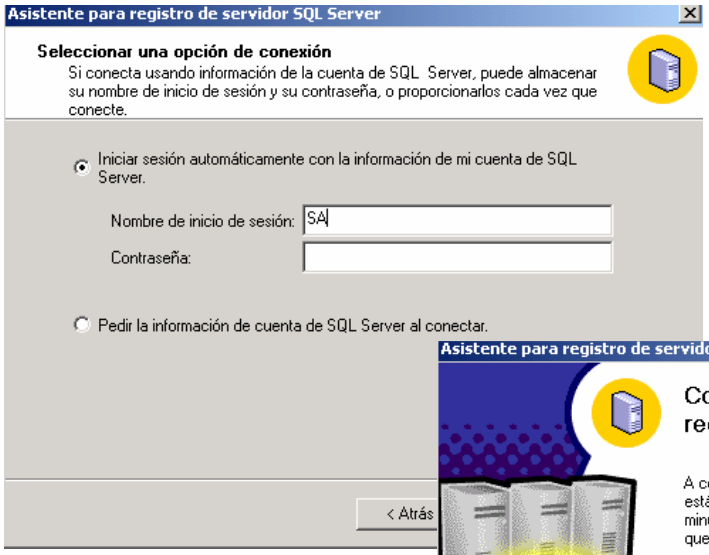
Aparecerá el asistente para registro de servidor SQL Server, pulsamos siguiente...



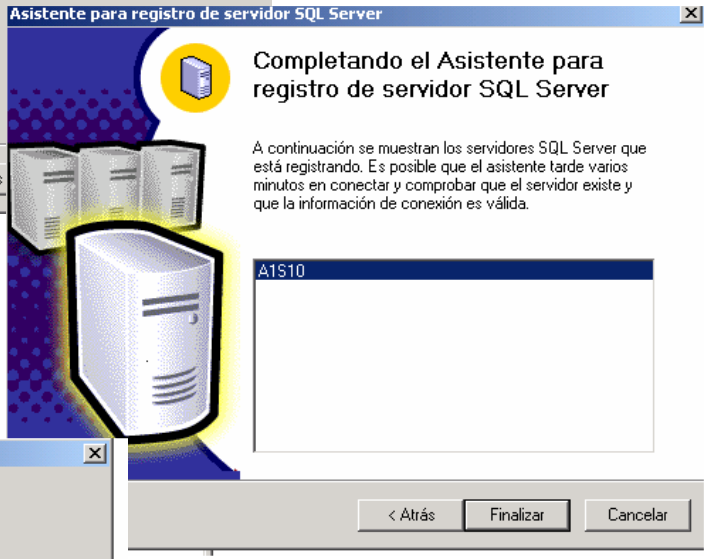
...y nos saldrá una pantalla que indicará los servidores que podemos incorporar a nuestro grupo de servidores, seleccionamos el / los que queramos y pulsamos agregar. Después pulsamos siguiente.

Aparecerá una pantalla que nos preguntará el tipo de conexión que usaremos, en este caso usaremos la segunda, porque trabajaremos con el usuario SA de Sql Server...

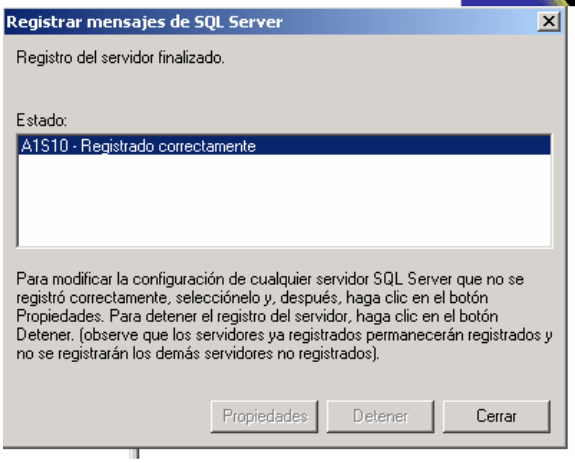




... posteriormente introducimos el login y la password. Pulsamos siguiente.



Nos indicará que esta completado el registro del servidor elegido. Pulsamos Finalizar.



Saldrá una pantalla en la que iremos viendo el proceso de registro del servidor elegido. Una vez finalizado este proceso, pulsamos cerrar y ya tendremos acceso a todas las bases de datos de ese servidor.

Una vez tengamos acceso a este servidor, para acceder a una tabla, procedimiento almacenado, vista etc. hemos de indicar primero el servidor, después la base de datos, luego el propietario y posteriormente la tabla, vista o lo que sea, conectandonos desde el servidor importado, para acceder al nuevo.
Para poder utilizarlo, tendremos que utilizar el procedimiento almacenado
Sp_addlinkedservers 'SERVIDOR'
Select * from Servidor.BBDD.propietario.Tabla / PA / Vista etc.
Select * from serra.hospital.dbo.emp
Para ver nuestra lista de servidores:

```
use master
go
select * from sysservers
```

- **OpenRowSet**

Accede a una vista, tabla, procedimiento almacenado de una base de datos perteneciente a otro servidor, sin necesidad de que el servidor esté vinculado a nuestro servidor.
Sintaxis:
OpenRowSet (‘TipoProveedor’,Servidor, IdUsuario, Password, Base de datos, “Consulta”)

Ejemplo:

```
use Northwind
GO
select s.* from
openrowset ('SQLOLEDB','A1S02';'sa'; "
,'Select * from Northwind.Dbo.Orders') as s


use Hospital
GO
Select H.emp_no,H.apellido,H.Salario,H.Dept_no from
openRowset ('SQLOLEDB','SERRA';'sa';"
,'Select * from Hospital.dbo.emp') as H
```

CONSULTAS SQL – XML

Estructura de un documento XML:

```
<root>
<Elemento1A>
        <subelemento1A>
                <subelemento1B>
                </subelemento1B atributo1 atributo2>
        </subelemento1A>
</Elemento1A>
</root>
```

Creación de un documento XML a partir de las consultas con SQL

- No se puede utilizar Group by, para ello es necesario utilizar un parche.
- Las únicas palabras clave adicionales que le enviamos a SQL Server son FOR XML AUTO al final de una instrucción SELECT común.
- Vista en el analizador de consultas, la salida muestra una sola columna con un nombre extravagante: XML seguido de una expresión tipo GUID (identificador global único).
- Toda la salida va a una sola línea, pero aun así la cuenta de filas afectadas indica 8 filas. La línea de salida es muy larga e incluye todo el conjunto de resultados.
- Todavía no está completa la compatibilidad de SQL con XML, por eso muchas cosas fallan.

```
Select apellido, salario, dept_no from emp for xml auto
```

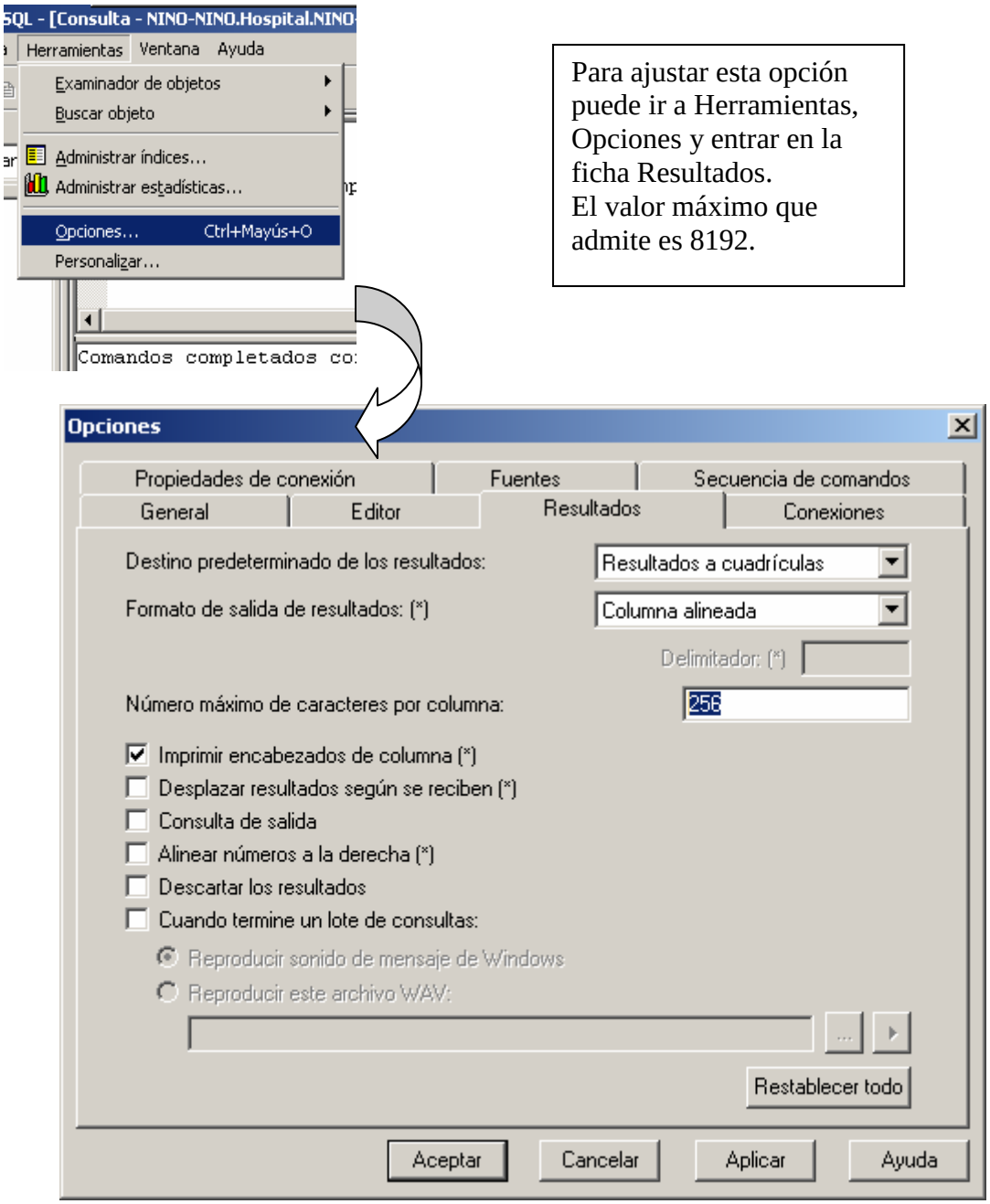
Documento XML generado:

- El documento generado por SQL no es completo, un documento SQL debe tener una única raiz, con lo cual para que el documento que nosotros recuperamos sea xml, debemos añadirle una raiz, entonces podremos verlo en el internet explorer.

```
<ListaEmpleados>
<emp apellido="SANCHEZ"
salario="104000"
dept_no="20"/>

<emp apellido="ARROYO"
salario="208000"
dept_no="30"/>

<emp apellido="SALA"
salario="162500"
dept_no="30"/>

</ListaEmpleados>
```

PRECAUCIÓN

Ajuste la cantidad máxima de caracteres por columna en el analizador de consultas a un valor mayor que 256, porque sino algunos de los valores devueltos en la consulta se truncarán.



En el ejemplo anterior, para la salida XML seleccionamos el modo AUTO .
SQL Server admite los modos siguientes:

- RAW : produce un flujo XML con elementos independientes para cada fila, donde cada elemento recibe el nombre row (independientemente del nombre de la tabla).
- AUTO : produce un flujo XML con elementos independientes para cada fila, donde cada elemento recibe el nombre de la tabla de la que se extrajeron los datos.

Tambien tenemos la opción de usar, **solamente en el auto**, la opción de Elements, que nos crea un documento XML pero separando cada uno de los campos que nosotros utilizamos.

```
Select apellido, salario, dept_no from emp for xml auto, elements
```

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B
--------------------------------------------------------------------------------
-----------------
<emp>
        <apellido>SANCHEZ</apellido>
        <salario>104000</salario>
        <dept_no>20</dept_no>
</emp>
<emp>
        <apellido>ARROYO</apellido>
        <salario>208000</salario>
        <dept_no>30</dept_no>
</emp>
<emp>
        <apellido>SALA</apellido>
        <salario>162500</salario>
        <dept_no>30</dept_no>
</emp>
<emp>
        <apellido>JIMENEZ</apellido>
        <salario>386750</salario>
        <dept_no>20</dept_no>
</emp>
.............

(16 filas afectadas)
```

Sin embargo, si la instrucción SELECT lee datos de más de una tabla hay una gran diferencia entre cómo trabajan los modos RAW y AUTO.

```
select
emp_no as [Numero]
,apellido as Apellido
,salario as Salario
,Departamento.dnombre as [NombreDepartamento] from emp as Empleados
inner join dept as Departamento
on Empleados.dept_no = Departamento.dept_no
for xml auto
```

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B
--------------------------------------------------------------------------------
-----------------
<Empleados Numero="7369" Apellido="SANCHEZ" Salario="104000">
        <Departamento NombreDepartamento="INVESTIGACION"/>
</Empleados>
<Empleados Numero="7499" Apellido="ARROYO" Salario="208000">
        <Departamento NombreDepartamento="Ventas"/>
</Empleados>
<Empleados Numero="7521" Apellido="SALA" Salario="162500">
        <Departamento NombreDepartamento="Ventas"/>
</Empleados>
<Empleados Numero="7566" Apellido="JIMENEZ" Salario="386750">
        <Departamento NombreDepartamento="INVESTIGACION"/>
</Empleados>
<Empleados Numero="7654" Apellido="MARTIN" Salario="182000">
        <Departamento NombreDepartamento="Ventas"/>
</Empleados>
...................

 (16 filas afectadas)
```

```
select
emp_no as [Numero]
,apellido as Apellido
,salario as Salario
,Departamento.dnombre as [NombreDepartamento] from emp as Empleados
inner join dept as Departamento
on Empleados.dept_no = Departamento.dept_no
for xml raw
```

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B
--------------------------------------------------------------------------------
<row apellido="SANCHEZ" salario="104000" dept_no="20"/>
<row apellido="Nino" salario="454" dept_no="10"/>
<row apellido="ARROYO" salario="208000" dept_no="30"/>
<row apellido="SALA" salario="162500" dept_no="30"/>
<row apellido="JIMENEZ" salario="386750" dept_no="20"/>
<row apellido="MARTIN" salario="182000" dept_no="30"/>
<row apellido="NEGRO" salario="370500" dept_no="30"/>
<row apellido="CEREZO" salario="318500" dept_no="10"/>
<row apellido="GIL" salario="390000" dept_no="20"/>
<row apellido="REY" salario="650000" dept_no="10"/>
<row apellido="TOVAR" salario="195000" dept_no="30"/>
<row apellido="ALONSO" salario="143000" dept_no="20"/>
<row apellido="JIMENO" salario="123500" dept_no="30"/>
<row apellido="FERNANDEZ" salario="390000" dept_no="20"/>
<row apellido="MUÑOZ" salario="169000" dept_no="10"/>

(15 filas afectadas)
```

La diferencia entre las dos sentencias es que auto crea el documento de forma jerarquica, mientras que raw crea el documento con elementos unicos y resuelve todas las columnas como atributos, independientemente de que sean de otra tabla.

XMLDATA

- Nos devuelve lo mismo que las otras sentencias normales, pero con la diferencia que nos devuelve un informe de cómo está echo el esquema del documento XML creado.

```
select
emp_no as [Numero]
,apellido as Apellido
,salario as Salario
,Departamento.dnombre as [Nombre] from emp as Empleados
inner join dept as Departamento
on Empleados.dept_no = Departamento.dept_no
for xml auto,xmldata
```

```
XML_F52E2B61-18A1-11d1-B105-00805F49916B
--------------------------------------------------------------------------------
------------------
<Schema name="Schema2" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-
microsoft-com:datatypes">
       <ElementType name="Empleados" content="eltOnly" model="closed" order="many">
              <element type="Departamento" maxOccurs="*"/>
                     <AttributeType name="Numero" dt:type="i4"/>
                     <AttributeType name="Apellido" dt:type="string"/>
                     <AttributeType name="Salario" dt:type="i4"/>
                     <attribute type="Numero"/>
                     <attribute type="Apellido"/>
                     <attribute type="Salario"/>
              </ElementType>
       <ElementType name="Departamento" content="empty" model="closed">
                     <AttributeType name="Nombre" dt:type="string"/>
                     <attribute type="Nombre"/>
       </ElementType>
</Schema>
```

Recuperación de datos en documentos XML

- **sp_xml_preparedocument:** Prepara SQL para poder utilizar un documento XML.

sp_xml_preparedocument (int output, Documento XML)

int: Es un integer que se asocia internamente al documento XML para poder utilizarlo con la instrucción Openxml, ya que la función necesita un integer, no el documento.

- **sp_xml_removedocument:** Elimina los residuos que ha dejado el documento XML en la memoria al utilizar sp_xml_preparedocument.

sp_xml_removedocument int output

int: Es el integer asociado al documento

OPENXML: Se utiliza para recuperar los datos de un documento XML, recorre la estructura del documento XML devolviendo lo que queramos.

OPENXML(int, RutaXML,Tipo Búsqueda)

Búsqueda:

0 Por defecto, el *mapping* (la traducción de los valores a XML) se basa en el uso de atributos.
1 Usar *mapping* basado en atributos.
2 Usar *mapping* basado en elementos.
3 Puede combinar *mapping* basado en elementos y en atributos.

Utilizamos WITH para sacar los datos dentro de la ruta XML que le hemos proporcionado.El nombre de los elementos que voy a recuperar con With, es el nombre de los elementos que voy a sacar del documento XML, deben coincidir mayúsculas y minúsculas y en los tipos de datos declarados.

OPENXML(int, RutaXML,Tipo Búsqueda) with (Atributo|Elemento)

Ejemplo para recuperar los datos de un documento XML:

Recuperar los datos a partir de los elementos: Me posiciono en Clientes y recupero los elementos ID y NOMBRE

```
declare @xml varchar(1000)
Declare @doc int
set @xml='
<principal>
        <clientes>
                <id>1</id>
                <Nombre>Miguel Egea</Nombre>
        </clientes>
        <clientes>
                <id>2</id>
                <Nombre>Jose Antonio Herrera</Nombre>
        </clientes>
        <clientes>
                <id>3</id>
                <Nombre>Eladio Rincon</Nombre>
        </clientes>
</principal>'
exec sp_xml_preparedocument @doc output,@xml
declare @Tabla table (id Int, Nombre Varchar(20))
insert into @tabla
select * from openxml(@doc,'principal/clientes',2)
WITH (id int
,Nombre varchar(20))
exec sp_xml_removedocument @doc output
select * from @tabla
```

Ejemplo2:

Ahora voy a buscar por los atributos:

```
declare @ixml int
declare @xml nvarchar(500)
set @xml = '
<ListaEmpleados>
        <emp apellido="SANCHEZ"
        salario="104000"
        dept_no="20"/>

        <emp apellido="ARROYO"
        salario="208000"
        dept_no="30"/>

        <emp apellido="SALA"
        salario="162500"
        dept_no="30"/>
```

```
        <emp apellido="REY"
        salario="650000"
        dept_no="10"/>
</ListaEmpleados>'
EXEC sp_xml_preparedocument @ixml output,@xml
SELECT    *
FROM OPENXML (@iXML, '/ListaEmpleados/emp',1)
        WITH (apellido  nvarchar(30)
        ,salario int
        ,dept_no int)
exec sp_xml_removedocument @ixml output
```

EJEMPLO 3:

Recuperar tanto elementos como atributos.

```
declare @ixml int
declare @xml nvarchar(2000)
set @xml = '
<RAIZ>
        <Empleados Numero="7369" Apellido="SANCHEZ" Salario="104000">
                <Departamento>INVESTIGACION</Departamento>
        </Empleados>
        <Empleados Numero="7499" Apellido="ARROYO" Salario="208000">
                <Departamento>Ventas</Departamento>
        </Empleados>
        <Empleados Numero="7521" Apellido="SALA" Salario="162500">
                <Departamento>Ventas</Departamento>
        </Empleados>
        <Empleados Numero="7782" Apellido="CEREZO" Salario="318500">
                <Departamento>CONTABILIDAD</Departamento>
        </Empleados>
        <Empleados Numero="7839" Apellido="REY" Salario="650000">
                <Departamento>CONTABILIDAD</Departamento>
        </Empleados>
</RAIZ>'
EXEC sp_xml_preparedocument @ixml output,@xml
insert into MiTabla
SELECT    *
FROM OPENXML (@iXML, '/RAIZ/Empleados',3)
        WITH (Numero  int
        ,Apellido nvarchar(30)
        ,Salario int
        ,Departamento nvarchar(40))
select * from MiTabla
```

Debo crear primero la tabla dónde voy a insertar los datos.

```
create table AtributosElementos
(Numero int
,Apellido nvarchar(50)
,salario int
,Departamento nvarchar(50))
```

Esto es para realizar consultas en el documento y poder crear un patrón de búsqueda.
Solamente voy a mostrar los Departamentos nombre sea Ventas.

```
print 'CONSULTA 2'
insert into #nino
SELECT    *
FROM OPENXML (@iXML, '/RAIZ/Empleados[Departamento=Ventas]',3)
WITH (Nombre  nvarchar(30))
exec sp_xml_removedocument @xml
```

Ahora voy a buscar por atributos, solamente mostraré los que tengan un salario mayor de 200000.

```
insert into #nino
SELECT    *
FROM OPENXML (@iXML, '/RAIZ/Empleados[@salario > 200000]',3)
WITH (Nombre  nvarchar(30))
exec sp_xml_removedocument @xml
```

Realizar un documento XML que muestre los datos de los jugadores del Real Madrid y del Barcelona, una vez realizada esta acción, distribuir los jugadores en tablas, dependiendo de su posición.

Optimizador de Consultas

El rendimiento de una consulta de base de datos puede determinarse si se utiliza la instrucción SET para habilitar las opciones SHOWPLAN, STATISTICS IO, STATISTICS TIME y STATISTICS PROFILE.

- SHOWPLAN describe el método elegido por el optimizador de consultas de SQL Server para recuperar los datos. Para obtener más información, consulte "SET SHOWPLAN_ALL" en *Referencia de Microsoft SQL Server Transact-SQL y herramientas*.
- STATISTICS IO muestra información acerca del número de exploraciones, lecturas lógicas (páginas a las que se tiene acceso en caché) y lecturas físicas (número de veces que se tuvo acceso al disco) para cada una de las tablas mencionadas en la instrucción. Para obtener más información, consulte "SET STATISTICS IO" en *Referencia de Microsoft SQL Server Transact-SQL y herramientas*.
- STATISTICS TIME muestra el tiempo (en milisegundos) necesario para analizar, compilar y ejecutar una consulta. Para obtener más información, consulte "SET STATISTICS TIME" en *Referencia de Microsoft SQL Server Transact-SQL y herramientas*.
- STATISTICS PROFILE muestra un conjunto de resultados tras cada consulta ejecutada que representa un perfil de la ejecución de la consulta. Para obtener más información, consulte "SET STATISTICS PROFILE" en *Referencia de Microsoft SQL Server Transact-SQL y herramientas*.

En el Analizador de consultas de SQL Server puede activar también la opción de **plan de ejecución gráfico** para ver una representación gráfica de cómo SQL Server recupera los datos.

La información recopilada por estas herramientas le permiten determinar cómo ejecuta una consulta el optimizador de consultas de SQL Server y qué índices se están utilizando. Con esta información puede determinar si es posible mejorar el rendimiento al volver a escribir la consulta, cambiar los índices de las tablas o, quizás, modificar el diseño de la base de datos donde sea posible.

Microsoft SQL Server ofrece tres formas de presentar información acerca de cómo se desplaza por las tablas y utiliza los índices para tener acceso a los datos de una consulta:

- Mostrar gráficamente el plan de ejecución con el Analizador de consultas de SQL Server

  En el Analizador de consultas de SQL Server, haga clic en **Consulta** y seleccione **Mostrar el plan de ejecución**. Después de ejecutar una consulta, puede seleccionar la ficha **Plan de ejecución** para ver una representación gráfica del resultado del plan de ejecución. Para obtener más información, consulte "Mostrar gráficamente el plan de ejecución mediante el Analizador de consultas de SQL Server" en este volumen.

- SET SHOWPLAN_TEXT ON

  Después de ejecutar esta instrucción, SQL Server devuelve la información del plan de ejecución para cada consulta. Para obtener más información, consulte "SET SHOWPLAN_TEXT" en *Referencia de Microsoft SQL Server Transact-SQL y herramientas*.

- SET SHOWPLAN_ALL ON

  Esta instrucción es similar a SET SHOWPLAN_TEXT, excepto que el resultado se muestra en un formato conciso. Para obtener más información, consulte "SET SHOWPLAN_ALL" en *Referencia de Microsoft SQL Server Transact-SQL y herramientas*.

Cuando muestre el plan de ejecución, no se ejecutarán las instrucciones que envíe al servidor; en su lugar, SQL Server analiza la consulta y muestra cómo se habrían ejecutado las instrucciones como una serie de operadores.

Los siguientes iconos mostrados en el plan de ejecución gráfico representan los operadores físicos que SQL Server utiliza para ejecutar instrucciones.

Icono	**Operador físico**
	Aserción
	Búsqueda por marcador
	Eliminación de índice agrupado
	Inserción de índice agrupado
	Exploración de índice agrupado
	Búsqueda de índice agrupado
	Actualización de índice agrupado
	Contraer
	Cálculo escalar
	Concatenación
	Exploración de constantes
	Exploración de eliminados
	Filtro

	Raíz de coincidencia hash
	Equipo de coincidencia hash
	Eliminación de índice
	Inserción de índice
	Exploración de índice
	Búsqueda de índice
	Cola de índice
	Actualización de índice
	Exploración de insertados
	Exploración de filas del registro
	Combinación de mezcla
	Bucles anidados
	Paralelismo
	Exploración de tabla de parámetros
	Eliminación remota
	Inserción remota
	Consulta remota
	Exploración remota
	Actualización remota
	Cola de recuento de filas
	Secuencia
	Ordenación
	Agregado de secuencia

	Eliminación de tabla
	Inserción de tabla
	Exploración de tabla
	Cola de tabla
	Actualización de tabla
	Superior

	Instantánea

Leer la información del plan de ejecución gráfico

La información del plan de ejecución gráfico del Analizador de consultas de SQL Server se lee de derecha a izquierda y de arriba abajo. Se muestran todas las consultas del lote que se analizan, incluido el costo de cada consulta como un porcentaje del costo total del lote.

- Cada nodo de la estructura en árbol se representa como un icono que especifica el operador lógico y físico utilizado para ejecutar parte de la consulta o instrucción.
- Cada nodo está relacionado con un nodo primario. Todos los nodos que tienen el mismo nodo primario se dibujan en la misma columna. Las reglas con puntas de flecha conectan cada nodo con su nodo primario.
- Las operaciones recursivas se muestran con un símbolo de iteración.
- Los operadores se muestran como símbolos relacionados con un nodo primario específico.
- Cuando la consulta contiene varias instrucciones, se dibujan varios planes de ejecución de la consulta.
- Las partes de las estructuras en árbol están determinadas por el tipo de instrucción ejecutada.

Tipo de instrucción	Elemento de la estructura en árbol
Transact-SQL y procedimientos almacenados	Si la instrucción es un procedimiento almacenado o una instrucción Transact-SQL, se convertirá en la raíz de la estructura en árbol del plan de ejecución gráfico. El procedimiento almacenado puede contener varios procesos secundarios que representan las instrucciones a las que llama el procedimiento almacenado. Cada proceso secundario es un nodo o rama del árbol.
Lenguaje de tratamiento de datos (DML)	Si la instrucción analizada por el optimizador de consultas de SQL Server es una instrucción DML, como SELECT, INSERT, DELETE o UPDATE, la instrucción DML será la raíz del árbol. Las instrucciones DML pueden tener hasta dos nodos secundarios. El primer nodo secundario es el plan de ejecución de la instrucción DML. El segundo nodo secundario representa un desencadenador, tanto si se utiliza en la instrucción como si es utilizado por ella.
Condicional	El plan de ejecución gráfico divide las instrucciones condicionales como IF…ELSE (si la condición se cumple, entonces hacer lo siguiente; de lo contrario, ejecutar esta instrucción en su lugar) en tres nodos secundarios. La instrucción IF…ELSE es la raíz del árbol. La condición IF se convierte en un nodo del árbol secundario. Las condiciones THEN y ELSE se representan como bloques de instrucciones. Las instrucciones WHILE y DO-UNTIL se representan mediante un plan similar.
Operadores relacionales	Las operaciones que realiza el motor de consultas, como exploraciones de tablas, combinaciones y agregados, se representan como nodos del árbol.

Cada nodo muestra información sobre herramientas cuando el cursor lo señala. La información sobre herramientas puede incluir:

- El operador físico (**Operación física**) utilizado, como una Combinación hash o Bucles anidados. Los operadores físicos mostrados en color rojo indican que el optimizador de consultas ha emitido una advertencia, como la falta de estadísticas de columna o de predicados de combinación. Esto puede hacer que el optimizador de consultas elija un plan de consulta menos eficaz que el esperado. Para obtener más información acerca de las estadísticas de columnas, consulte "Información estadística" en el *Libro guía del programador de bases de datos de Microsoft SQL Server*. El plan de ejecución gráfico sugiere una solución, como la creación o actualización de estadísticas, o la creación de un índice. Las estadísticas de columnas y los índices que faltan pueden crearse o actualizarse inmediatamente mediante los menús contextuales del Analizador de consultas de SQL Server.
- El operador lógico (**Operación lógica**) que coincide con el operador físico, como el operador de combinación (Join). El operador lógico, si es diferente que el operador físico, se muestra después del operador físico en la parte superior de la información de herramientas separado por una barra diagonal (/).
- El número de filas (**Recuento de filas**) que devuelve el operador.

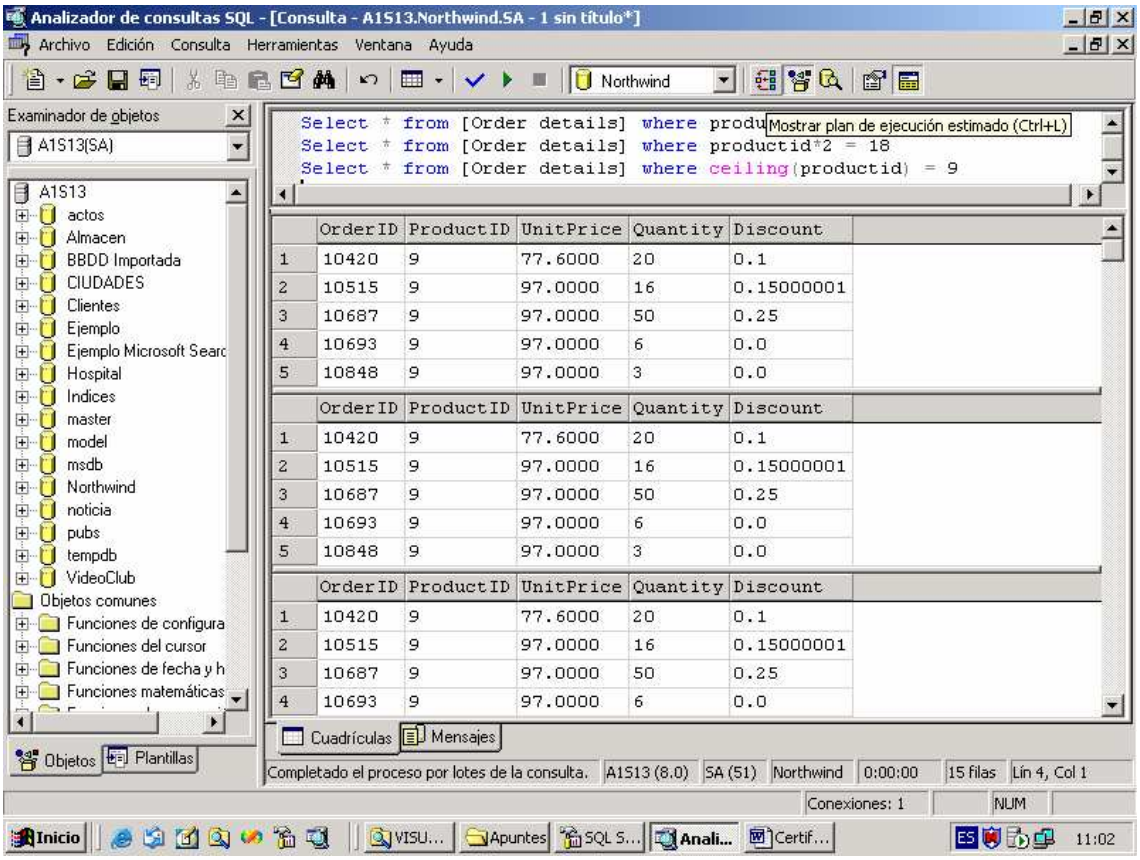
- El tamaño estimado de la fila (**Tamaño de fila estimado**) que devuelve el operador.
- El costo estimado (**Costo de E/S**) de toda la actividad de E/S de la operación. Este valor debe ser el menor posible.
- El costo estimado de toda la actividad de la CPU (**Costo de CPU**) de la operación.
- El número de veces que se ha ejecutado la operación (**Número de ejecuciones**) durante la consulta.
- El costo del optimizador de consultas (**Costo**) al ejecutar esta operación, incluyendo el costo de esta operación como un porcentaje del costo total de la consulta. Este valor debe ser el menor posible, ya que el motor de consultas selecciona la operación más eficiente para realizar la consulta o ejecutar la instrucción.
- El costo total del optimizador de consultas (**Costo de subárbol**) al ejecutar esta operación y todas las operaciones del mismo subárbol anteriores a ésta.
- Los predicados y parámetros (**Argumento**) utilizados por la consulta.

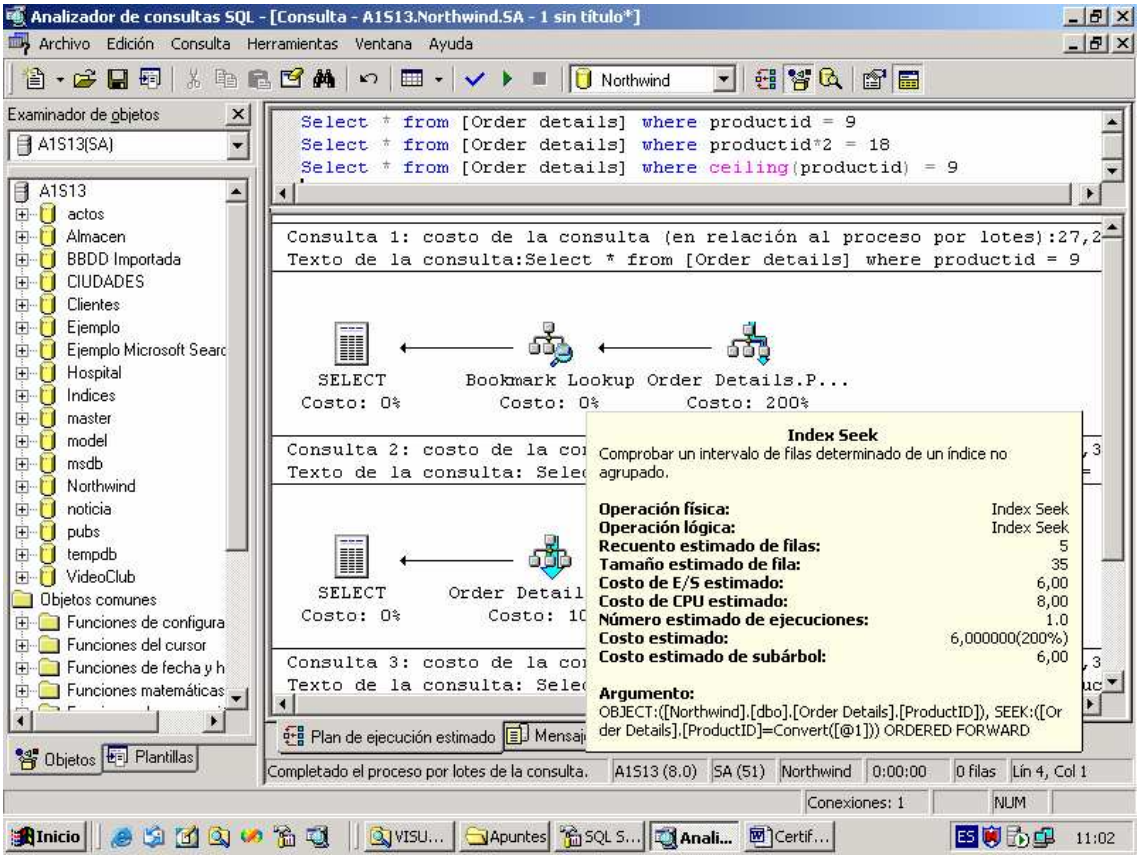
Cuando realizamos una consulta se ponen en marcha una serie de procesos aun que sea una consulta que pueda resultar muy sencilla:

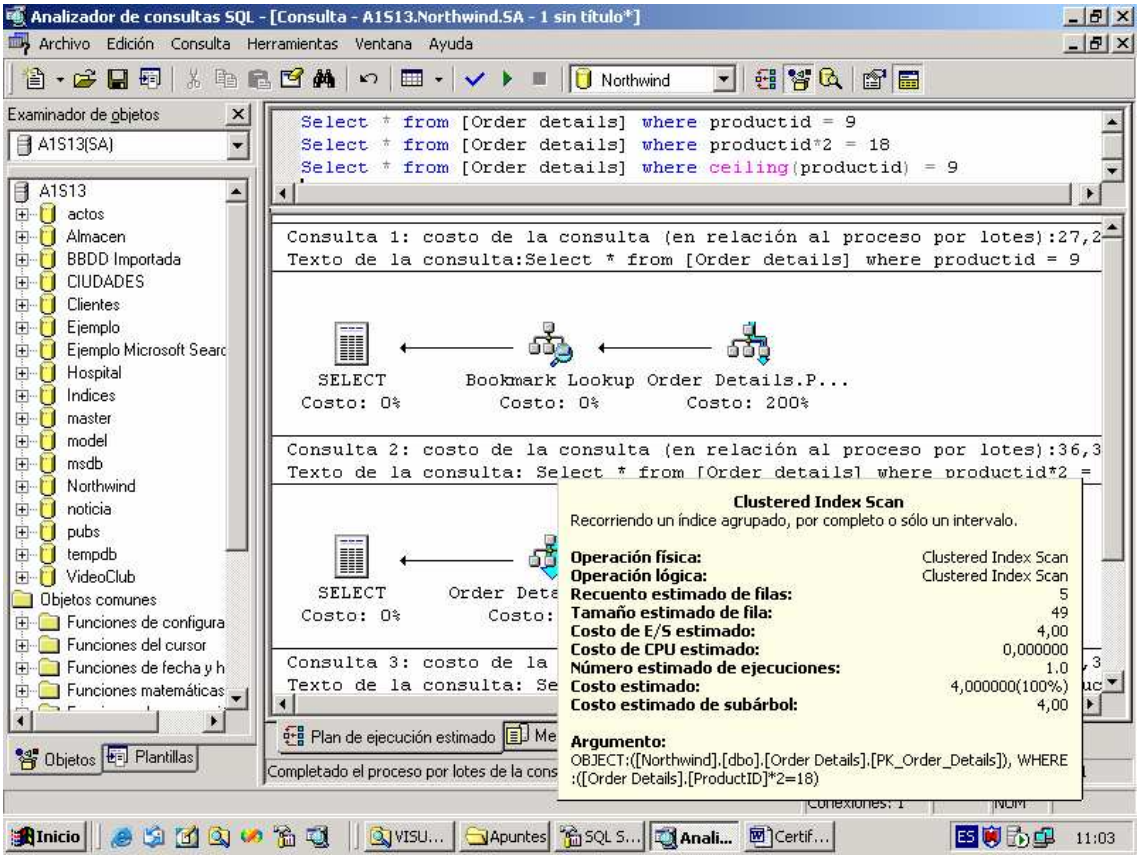
1. Analisis: Se analiza la sintaxis de la consulta para comprobar si es correcta.
2. Estandarización: Se modifica la consulta estandarizándo las subconsultas si las tiene y eliminando partes de la consulta que se repitan.
3. Optimización: Se desarrollan una serie de pasos para averiguar que plan de los posibles que hay, es el mejor para realizar la consulta. De todos los pasos los que mas influyen son:
    - Análisis de la consulta: Evalua el número de filas que se van a procesar, para saber el nº de páginas de índices y datos leídos.
    - Selección de índices: Averigua si existe un índice, si existe un índice para las columnas que se incluyen en la consulta. Usando las estádisticas del índice, evalua la mejor forma de ejecutar la consulta, usando o no usando los índices.

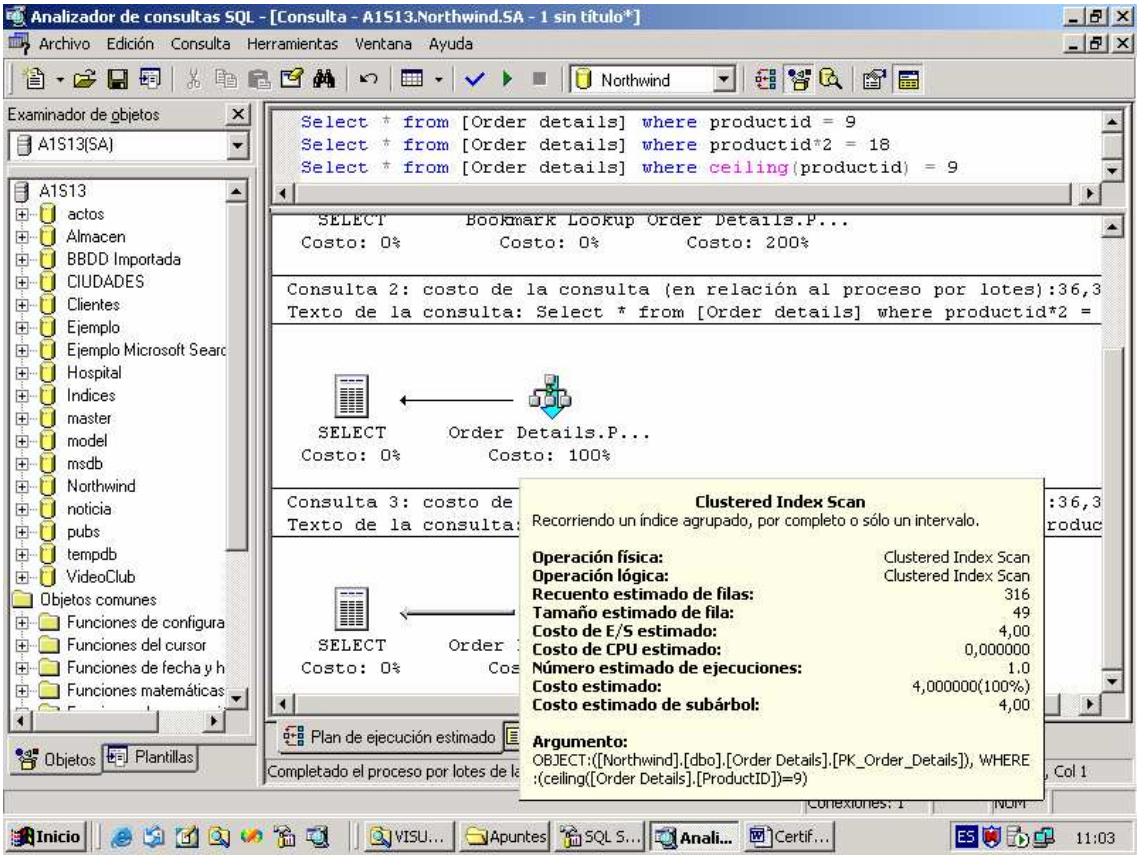
1. **Compilación:** Averigua si es mas sencillo usar índices para resolver la consulta o puede realizarse sin índices, consultando cada hoja de datos registro a registro una por una. A la hora de tener en cuenta una cosa u otra no tiene en cuenta el tiempo que tarde, sino los recursos que consuma. Una vez hecho esto genera el plan para realizar la consulta.
   Si por ejemplo hacemos:
   Select * from Emp where oficio = ‘Vendedor’
   Consulta como están realizados los índices.
   Tiene en cuenta como sea la consulta, si tiene comodines o no etc.
   Ejemplo Like ‘%dor’ No usará índices ya que debe buscar en todos los registros.
   Like ‘Ven%’ Usará índices. Porque los registros estarán mas localizados.
   Recorrerá todas las páginas de índices y buscará donde comienzan las palabras con ‘Ven’ irá a la hoja de datos.
   Mira tambienb si tiene índices y están a su disposición si los índices se incluyen en la consulta o no.
   Consulta una serie de estadísticas que se realizan sobre la tabla, estas estadísticas se generan una vez y hasta que no queramos no se generan de nuevo.

2. Optimización:






Analizador de consultas SQL - [Consulta - A1S13.Northwind.SA - 1 sin título*]
Archivo Edición Consulta Herramientas Ventana Ayuda
Northwind
Examinador de objetos
A1S13(SA)
A1S13
actos
Almacen
BBDD Importada
CIUDADES
Clientes
Ejemplo
Hospital
Indices
master
model
msdb
Northwind
noticia
pubs
tempdb
VideoClub
Objetos comunes
Select * from [Order details] where productid = 9
Select * from [Order details] where productid*2 = 18
Select * from [Order details] where ceiling(productid) = 9
Consulta 1: costo de la consulta (en relación al proceso por lotes):27,2
Texto de la consulta:Select * from [Order details] where productid = 9
SELECT
Costo: 0%
Bookmark Lookup
Costo: 0%
Order Details.P...
Costo: 200%
Consulta 2: costo de la consulta (en relación al proceso por lotes):36,3
Clustered Index Scan
Recorriendo un índice agrupado, por completo o sólo un intervalo.
Operación física: Clustered Index Scan
Operación lógica: Clustered Index Scan
Recuento estimado de filas: 5
Tamaño estimado de fila: 49
Costo de E/S estimado: 4,00
Costo de CPU estimado: 0,000000
Número estimado de ejecuciones: 1,0
Costo estimado: 4,000000(100%)
Costo estimado de subárbol: 4,00
Argumento:
OBJECT:([Northwind].[dbo].[Order Details].[PK_Order_Details]), WHERE :([Order Details].[ProductID]*2=18)
Plan de ejecución estimado
Completado el proceso por lotes de la cons
Inicio
11:03


Analizador de consultas SQL - [Consulta - A1S13.Northwind.SA - 1 sin título*]
Archivo Edición Consulta Herramientas Ventana Ayuda
Northwind
Examinador de objetos
A1S13(SA)
A1S13
actos
Almacen
BBDD Importada
CIUDADES
Clientes
Ejemplo
Hospital
Indices
master
model
msdb
Northwind
noticia
pubs
tempdb
VideoClub
Objetos comunes
Select * from [Order details] where productid = 9
Select * from [Order details] where productid*2 = 18
Select * from [Order details] where ceiling(productid) = 9
SELECT
Costo: 0%
Bookmark Lookup
Costo: 0%
Order Details.P...
Costo: 200%
Consulta 2: costo de la consulta (en relación al proceso por lotes):36,3
Texto de la consulta: Select * from [Order details] where productid*2 =
SELECT
Costo: 0%
Order Details.P...
Costo: 100%
Consulta 3: costo de
Texto de la consulta:
Clustered Index Scan
Recorriendo un índice agrupado, por completo o sólo un intervalo.
Operación física: Clustered Index Scan
Operación lógica: Clustered Index Scan
Recuento estimado de filas: 316
Tamaño estimado de fila: 49
Costo de E/S estimado: 4,00
Costo de CPU estimado: 0,000000
Número estimado de ejecuciones: 1.0
Costo estimado: 4,000000(100%)
Costo estimado de subárbol: 4,00
Argumento:
OBJECT:([Northwind].[dbo].[Order Details].[PK_Order_Details]), WHERE :(ceiling([Order Details].[ProductID])=9)
Plan de ejecución estimado
Completado el proceso por lotes de la
Inicio
11:03

```
select apellido, dnombre from emp
where detp.dept_no = emp.dept_no
```

Puede hacerlo de 2 formas:

Bucle:

Recorre la tabla departamentos mira todos los nºs de departamentos. Recorre emp y mira que coincida nº de dpto con nº de dpto de emp.

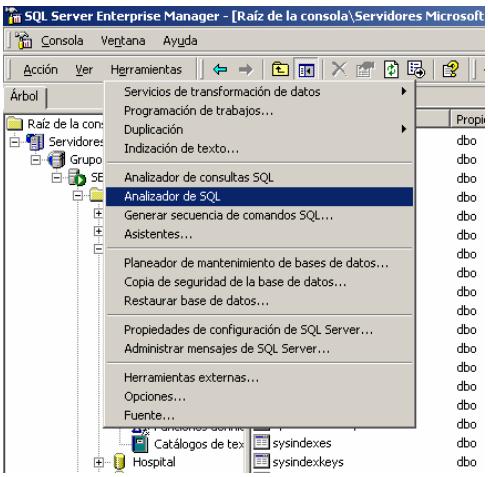
Hash:

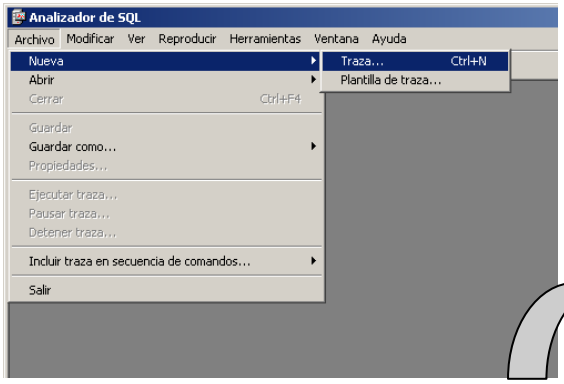
Es un calculo realizado sobre una columna indizada, por ej. un dni, divide cada dni entre 13 y dependiendo del resto agrupa cada dni en un determinado grupo dependiendo del resto. Se suele emplear hash cuando no hay un índice o los índices que hay no son válidos para la consulta.

Trazas

Las trazas son unas herramientas que nos permiten guardar la actividad de las bases de datos para luego poder sacar un informe del rendimiento que tienen. El informe del rendimiento será utilizado más tarde para optimizar las consultas

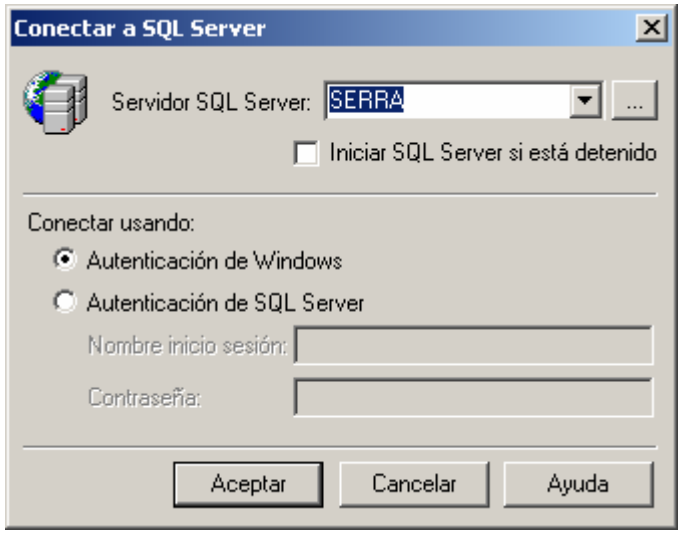
Para poder crear un archivo de trazas debemos seguir los siguientes pasos: Desde el administrador corporativo seleccionamos herramientas, Analizador de Consultas.



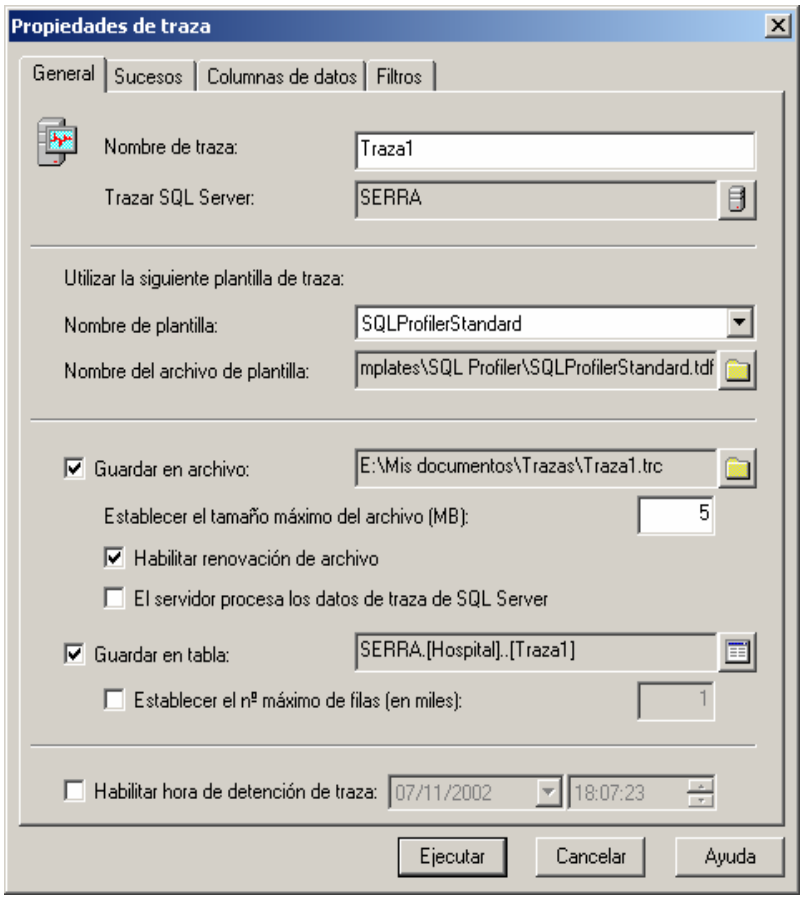


Entraremos en el analizador de SQL, desde dónde podremos crear una nueva traza. Para seleccionar las trazas debemos ir a archivo, Nuevo → Traza

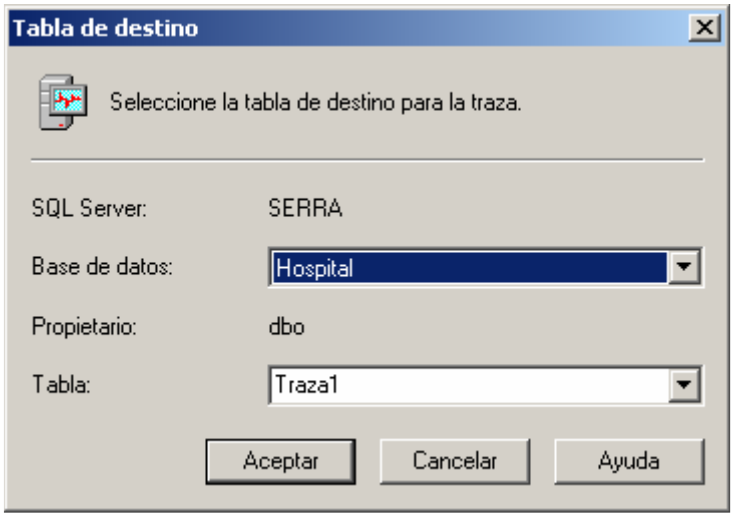
Nos pedirá que nos conectemos con el usuario que seamos, ya que es una parte diferente del administrador corporativo, igual que el analizador de consultas si entramos sin seleccionar ninguna tabla.



Esta es la pantalla inicial de una traza, en la que podremos seleccionar diferentes opciones para configurar nuestro archivo de registro.



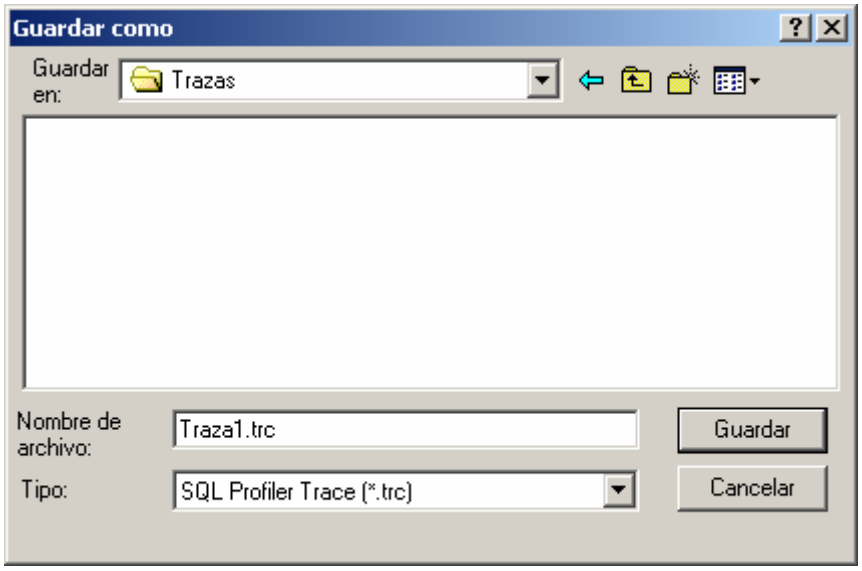
Una vez conectados con nuestro usuario, nos aparece una pantalla en la que debemos de rellenar algunos datos. Lo primero es darle nombre a la traza que vamos a crear. Las opciones que nos interesan son: Guardar en archivo y Guardar en Tabla.



Guardar en Tabla: El archivo de traza podremos vincularlo a una tabla para poder utilizarlo más tarde con el optimizador de indices.

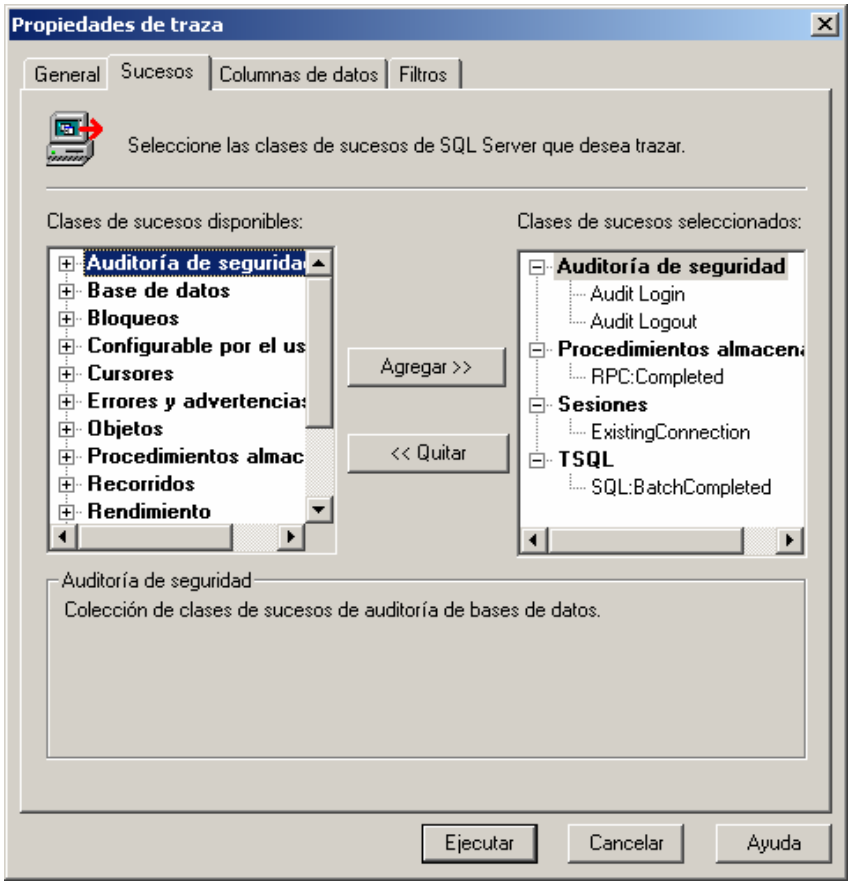
Habilitar hora de detención de traza: Podemos dejar una traza trabajando sobre unas determinadas operaciones dentro de la base de datos, con esta opción podemos programar a que hora paramos la traza para evaluar el movimiento de la base de datos.

Guardar en archivo: Nos ofrece la posibilidad de guardar la traza que vamos a crear en un archivo, que tendrá la extensión trc. Más tarde podremos recuperar este archivo de traza por si no queremos utilizarlo en ese momento y solamente queremos grabar una serie de movimientos dentro de nuestra base de datos.

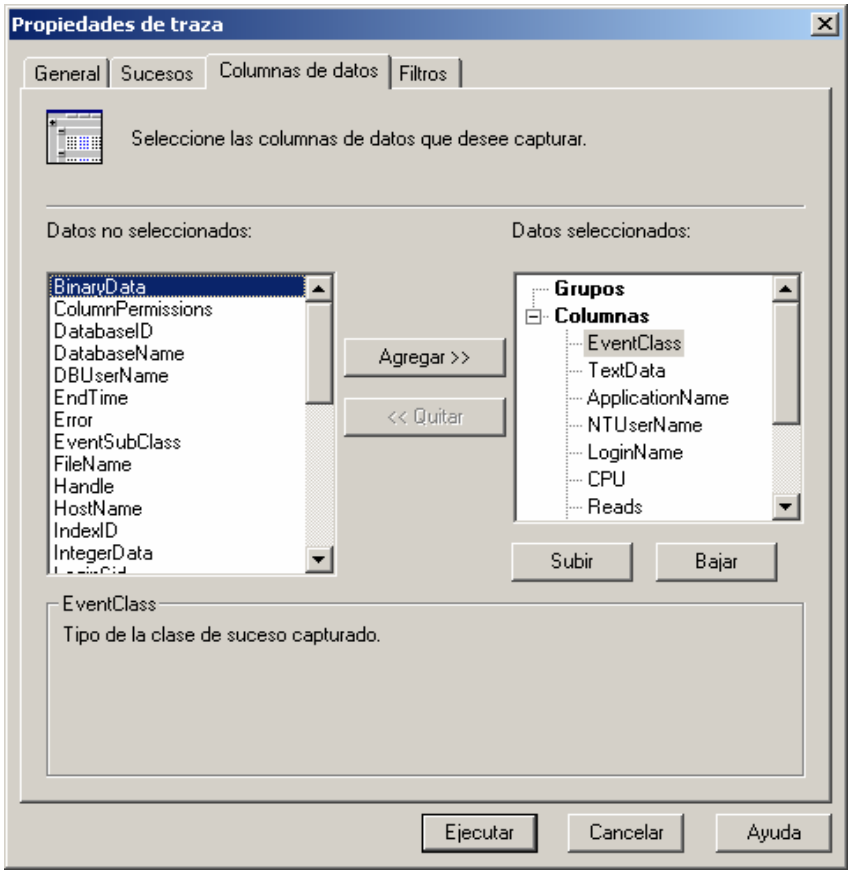


Nos guardará la traza dónde nosotros deseemos y con la extensión **.trc**

Esta traza podremos recuperarla más tarde.

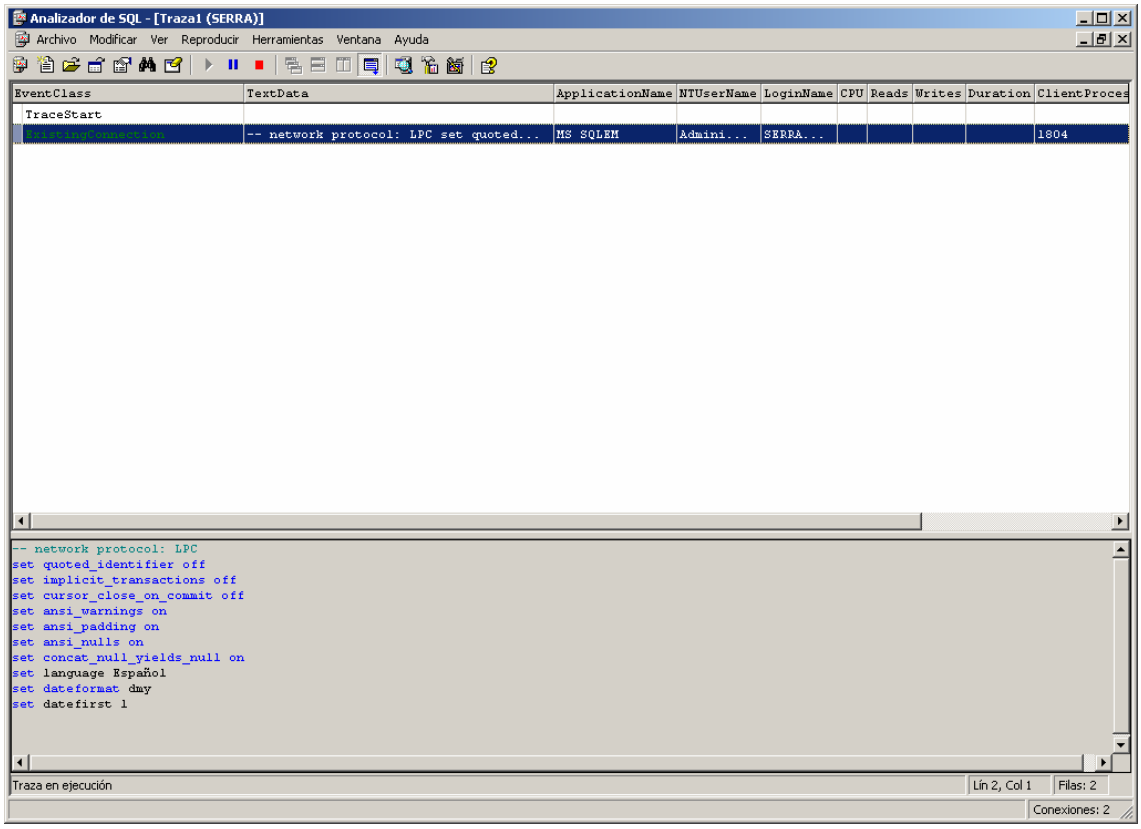


En esta pantalla podremos seleccionar los sucesos que queramos que SQL verifique con la traza. En la parte inferior nos muestra la definición de lo que vamos a Trazar. Seleccionaremos Rendimiento y Bases de datos.



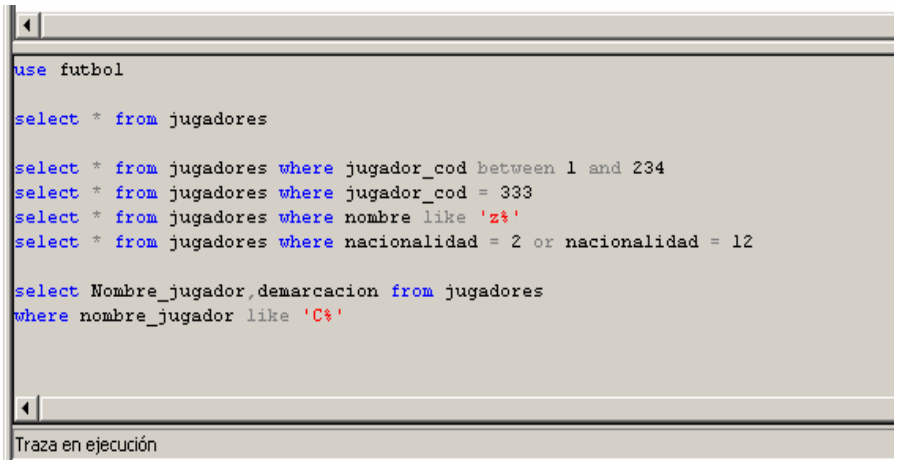
Esta pantalla nos permite seleccionar los diferentes campos que queremos registrar en nuestra traza, en la parte inferior nos muestra una pequeña definición de los datos que vamos a insertar.

Esta pantalla muestra una traza en ejecución, que podremos parar en todo momento o pausarla, una vez en ejecución ya podemos ir al analizador de consultas para escribir las consultas más utilizadas y capturar su valor dentro de la traza.



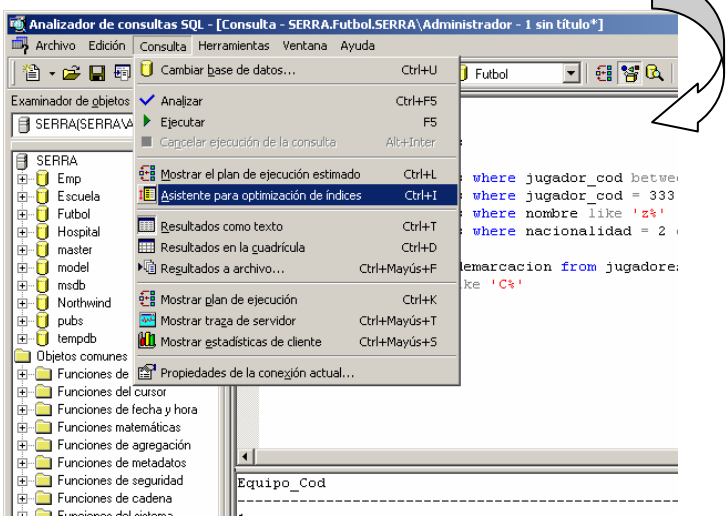
Con la traza en ejecución probaremos unas cuantas consultas en la base de datos que hemos seleccionado.

```
use futbol
GO
select * from jugadores

select * from jugadores where jugador_cod between 1 and 234
select * from jugadores where jugador_cod = 333
select * from jugadores where nombre like 'z%'
select * from jugadores where nacionalidad = 2 or nacionalidad = 12

select Nombre_jugador,demarcacion from jugadores
where nombre_jugador like 'C%'
```

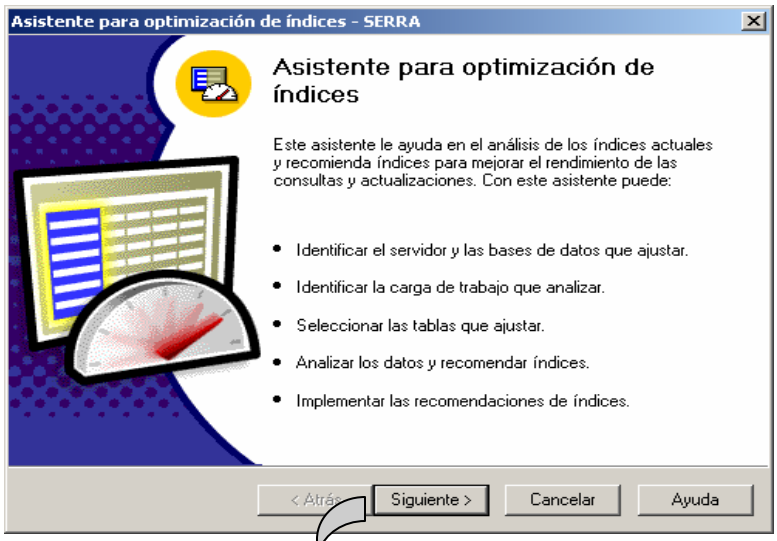


La traza captura el texto para después comprobar si necesitaríamos mejorar la estructura de la base de datos a partir de las consultas realizadas.

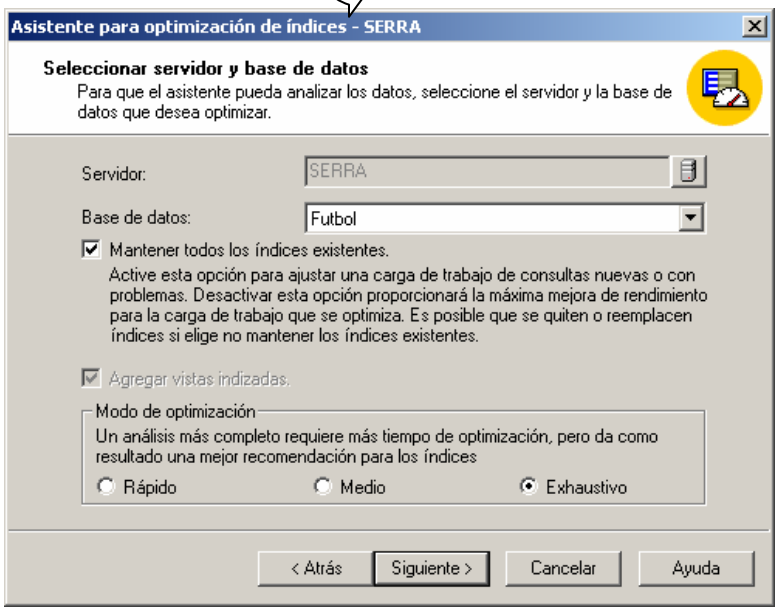
Una vez que he realizado las operaciones detengo la traza volviendo al Analizador de Consultas y pulsando sobre el botón Detener.



Realizado esto voy al asistente para optimización de indices para que me muestre lo que debería hacer con mi tabla a partir de las consultas que yo he realizado más frecuentemente.



Se abrirá un asistente para la optimización de indices, el cúal me guiará hasta encontrar la optimización de mi tabla para sacar mayor rendimiento a las consultas.

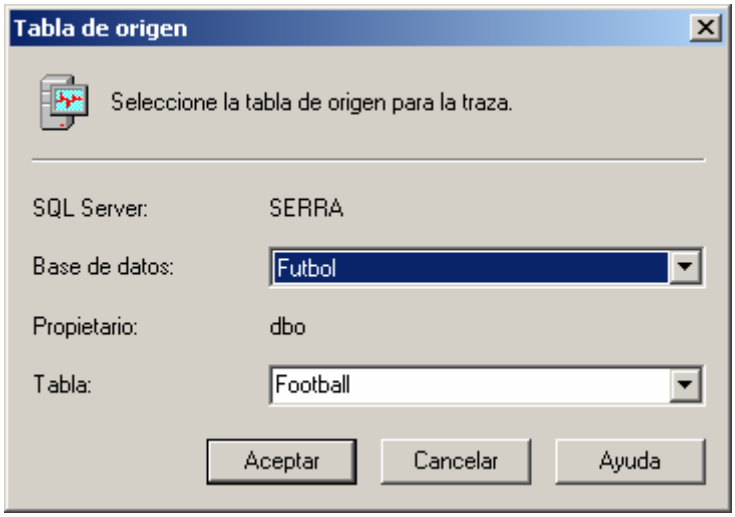


Se abrirá una ventana en la que estará mi servidor, la base de datos a optimizar y el tipo de análisis que quiero realizar sobre mi tabla. Seleccionaré exhaustivo.

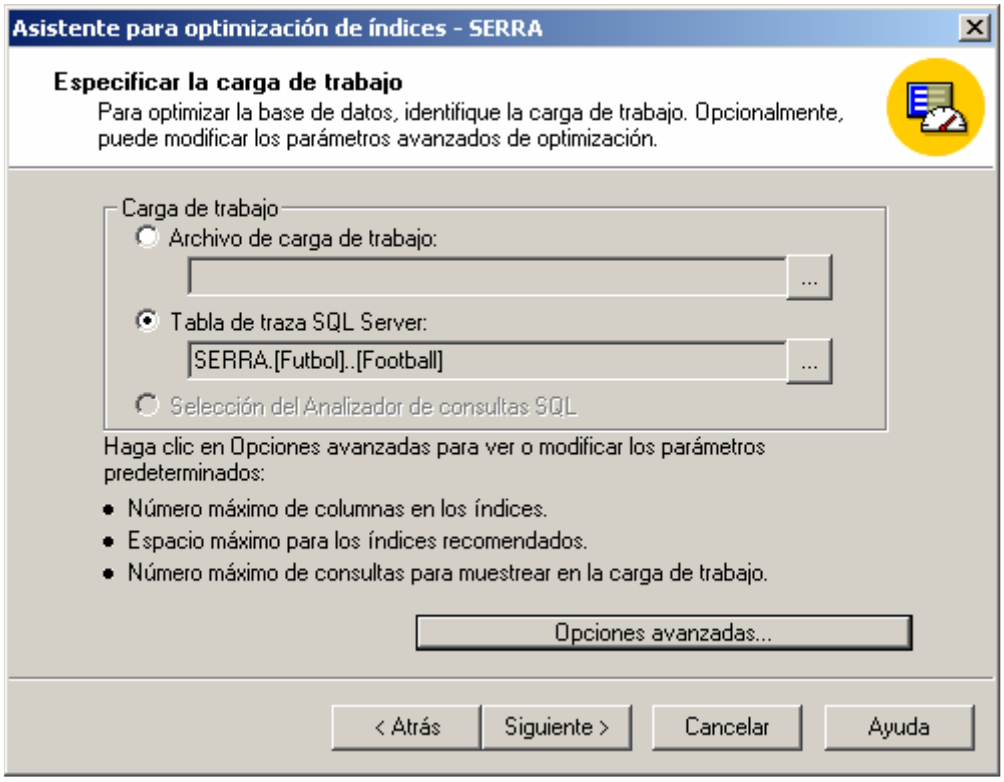
Al pulsar en siguiente aparecerá otra ventana en la que podré seleccionar dónde está la información para optimizar mi tabla, esta información deberé haberla escrito previamente mediante una traza.

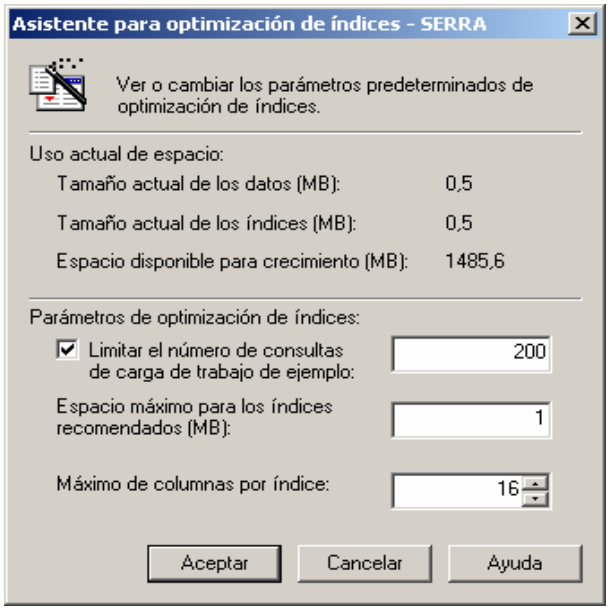
Archivo de carga de trabajo: Es un archivo de tipo.trc que podré importar para buscarlo y realizar el análisis desde ahí.

Tabla de traza SQL Server: Es una traza que he guardado en una tabla.

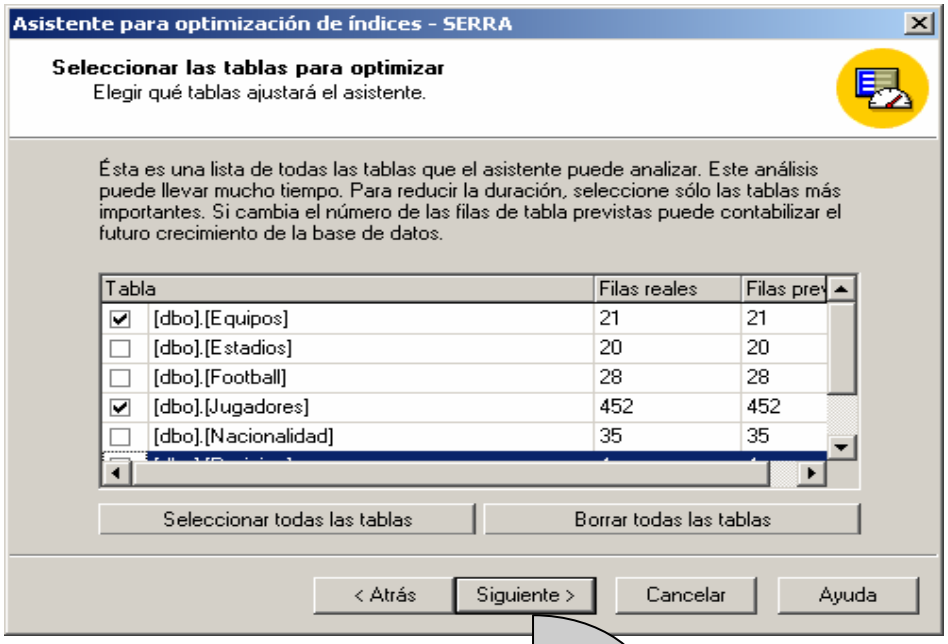


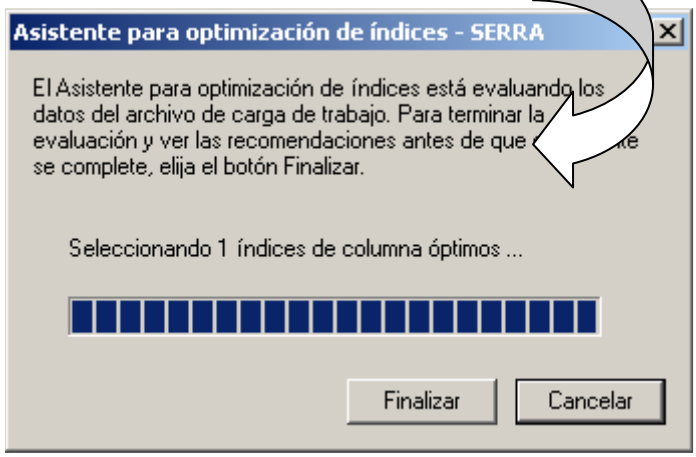
Selección del analizador de consultas: Una selección de texto que haya en el analizador de consultas para optimizar desde ahí.





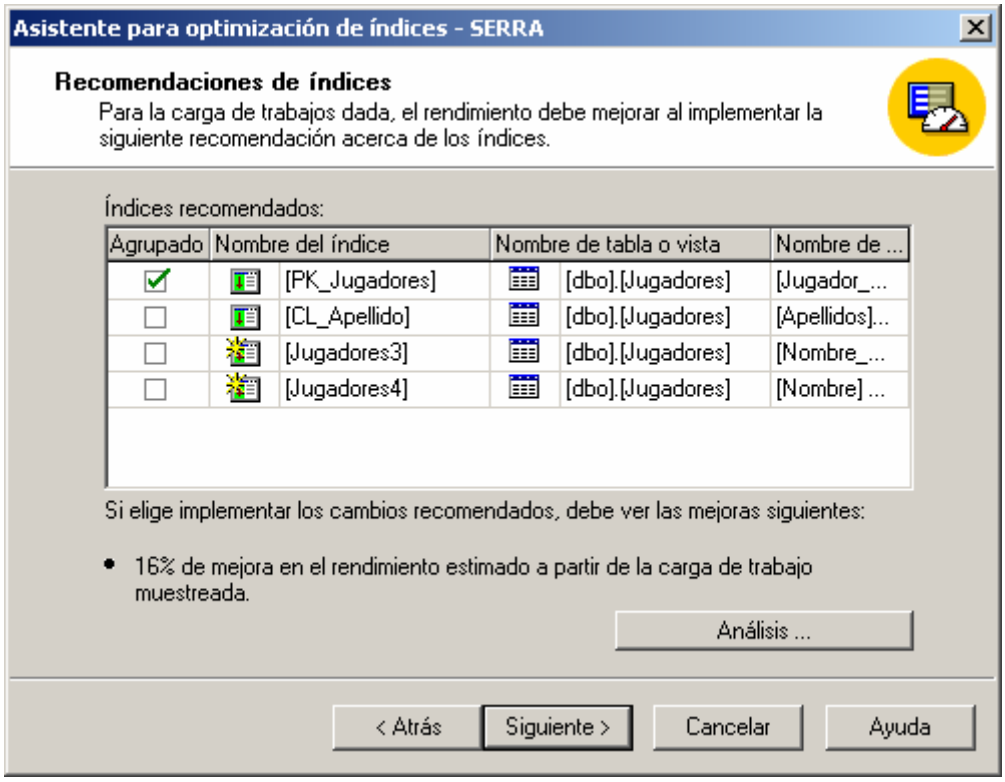
Pulsando sobre opciones avanzadas aparece una ventana del asistente en el que podremos seleccionar algunos ajustes para limitar el resultado que queremos en la configuración.

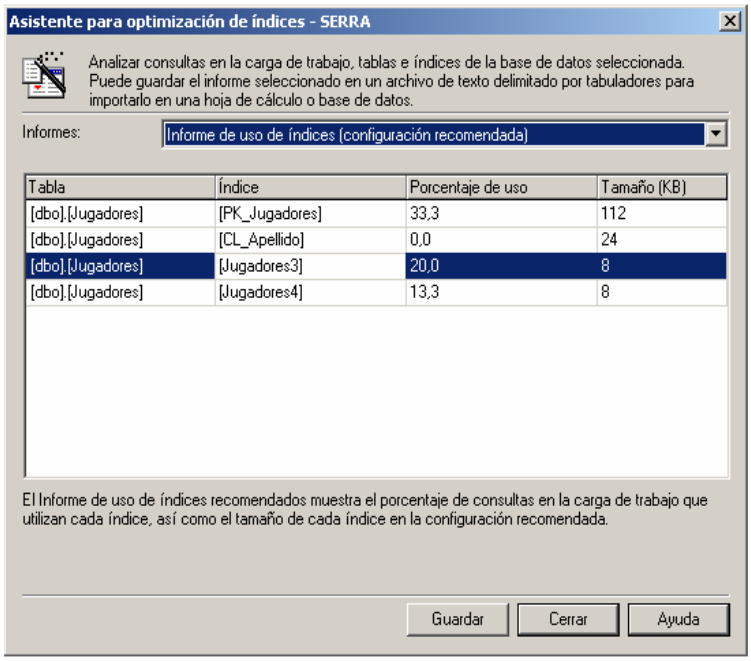




Una vez que pulsamos en siguiente, aparece una pantalla en la que podremos seleccionar las tablas sobre las que queremos realizar el análisis, debemos coger las tablas que hayamos usado para generar las consultas de la traza.

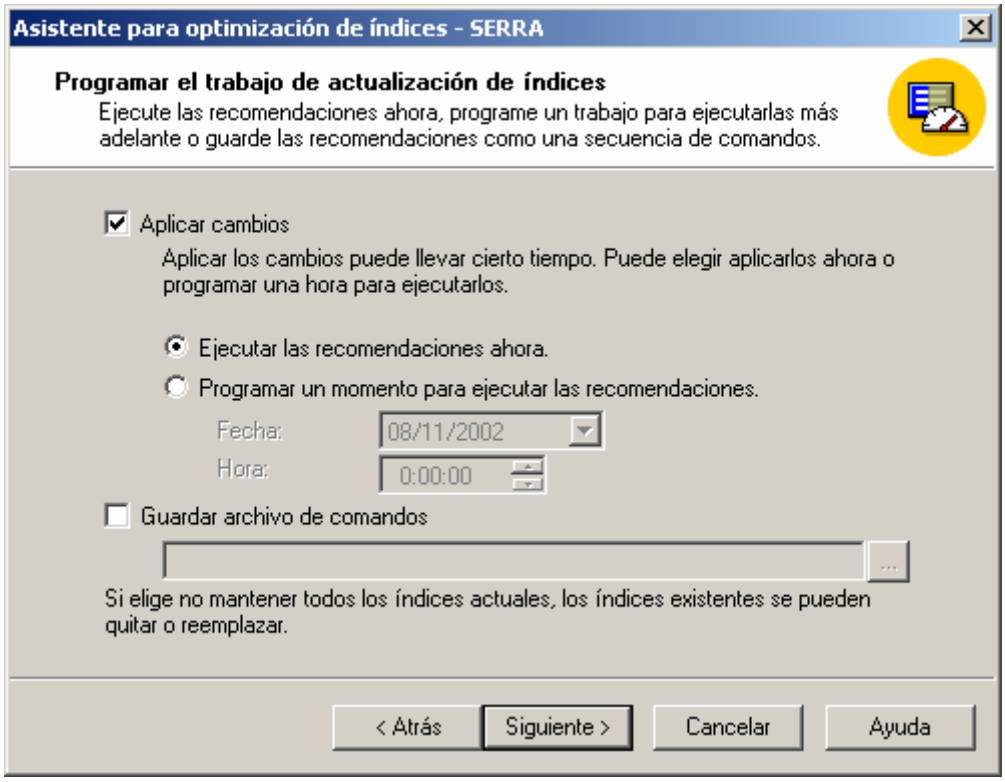
Una vez terminada la optimización nos devolverá el resultado que se ha estimado conveniente para poder optimizar las consultas y hacer más rapidamente las búsquedas.





Si pulsamos sobre análisis, veremos una pantalla dónde nos mostrará algunos datos sobre el uso de la tabla a partir de las consultas que hemos realizado. Podremos seleccionar entre varias opciones que nos mostrarán el rendimiento actual de la tabla y el óptimo.

Pulsando en siguiente podremos aplicar los cambios según ha estimado el analizador de consultas, se podrá seleccionar entre aplicar los cambios en este momento o programar el cambio, por si el servidor tiene mucho volumen en ese momento.



Consultas con tablas de sistema:

Muestra todas las columnas de una tabla determinada:

```
select c.Name, c.id, d.Name from syscolumns c, sysobjects d
where d.id = c.id and d.Name='emp'
```

```
sp_attach_db @dbname = 'Mi_Base_de_Datos',
  @filename1 = 'C:\Mi_base_de_datos_data.mdf',
  @filename2 = 'C:\Mi_base_de_datos_log.ldf'
```