

**Curso: 2012-2013**

# **TEORÍA DE (transact) SQL**

Pon tu nombre y apellidos  
Fecha:

**ÍNDICE**

1)Bases de datos utilizadas para los ejemplos: .....	6
a)VIDEOCLUBS GLOB-GUSTERS:.....	6
i)Esquema percibido:.....	6
ii)Grafo relacional: .....	6
iii)Sentencias SQL para la creación: .....	7
iv)Sentencias SQL para la inserción:.....	9
v)Sentencias SQL para la consulta de datos:.....	12
b)PROYECTALIA:.....	30
i)Esquema percibido:.....	30
ii)Grafo relacional: .....	30
iii)Sentencias SQL para la creación: .....	31
iv)Sentencias SQL para la inserción de datos: .....	33
v)Sentencias SQL para la consulta de datos:.....	34
2)Introducción.....	48
a)¿Qué es? .....	48
b)¿Para qué sirve? .....	48
c)Subtipos del lenguaje: DDL, DML, DCL. ....	48
d)Componentes.....	50
i)Comandos .....	50
ii)Claúsulas .....	51
iii)Operadores lógicos .....	51
iv)Operadores de comparación.....	51
v)Funciones de agregado.....	51
vi)Orden de ejecución de los comandos .....	51
3)Tipos de datos de SQL Server 2005 (se debe actualizar).....	52
a)Tipos de datos provisto pr SQL Server.....	52
b)Tipos de datos definidos por el usuario .....	53
4)Constantes: cómo se expresan las cadenas de caracteres, los enteros, las fechas, las horas,.....	56
5)Creación, modificación y eliminación de objetos (BD, Tablas, Vistas, Índices,...)	59
a)CREATE .....	59
i)Base de datos .....	59
ii)Tablas:.....	60
(1)Tipos de datos de las columnas .....	60
(2)Restricciones .....	61
(a)PRIMARY KEY .....	61
(b)UNIQUE .....	62
(c)NOT NULL .....	63
(d)FOREING KEY .....	63
(i)ON DELETE .....	65

(ii)ON UPDATE .....	66
(e)Propiedad IDENTITY .....	67
(3)Valores por defecto .....	68
(4)Crear una tabla a partir de otra.....	69
(5)Crear una tabla temporal .....	70
(6)Orden de creación atendiendo a las claves ajenas .....	71
iii)Vistas.....	71
iv)Índices .....	72
b)ALTER .....	75
i)Base de datos .....	75
ii)Tablas .....	75
(1)Añadir columnas .....	76
(2)Modificar columnas .....	76
(3)Borrar columnas .....	77
(4)Modificar restricciones .....	77
iii)Vistas.....	78
iv)Índices .....	79
c)DROP .....	80
i)Base de datos .....	80
ii)Tablas .....	80
(1)Eliminar columnas .....	80
(2)Eliminar restricciones.....	81
iii)Vistas.....	81
iv)Índices .....	81
6)Creación, modificación y eliminación de datos .....	82
a)INSERT .....	82
i)Inserción de filas .....	82
ii)Inserción individual de filas.....	82
iii)Inserción múltiple de filas .....	83
iv)Inserción de valores por posición. ....	84
v)Inserción de valores por nombre de columna.....	84
vi)Valores por defecto y valores nulos.....	84
b)UPDATE .....	85
i)Cambiar una columna de una fila .....	86
ii)Cambiar varias columnas de una fila.....	86
iii)Modificar datos en varias filas .....	86
iv)Uso de expresiones en la asignación .....	87
v)Valores por defecto y valores nulos.....	87
c)DELETE .....	87
d)TRUNCATE .....	88
7)Consultas: SELECT .....	90

a) Sintaxis de las consultas.....	90
i) La cláusula WHERE .....	91
ii) La cláusula GROUP BY .....	93
iii) La cláusula HAVING .....	93
iv) La cláusula ORDER BY .....	94
v) Funciones escalares para SELECT .....	95
b) Consultas sencillas sobre una tabla (o una vista) .....	96
i) Que muestre todos los atributos y todas las filas (*). .....	96
ii) Que muestre todos los atributos y filas seleccionadas (WHERE con operadores de relación, IN, BETWEEN, LIKE, IS NULL, ANY, ALL, EXISTS, ...). .....	96
iii) Que muestre algunos atributos y todas las filas (- ALL).....	99
iv) Que muestre algunos atributos y las filas no repetidas (DISTINCT) .....	100
v) Que muestre atributos y tablas con alias (AS) .....	100
vi) Que muestre algún campo como resultado de la operación de un atributo (columna calculada) .....	101
vii) Que muestre las primeras n filas (TOP, %). .....	101
viii) Que muestre las filas ordenadas por un atributo (ORDER BY, ASC, DESC).....	102
ix) Agrupar filas (GROUP BY, HAVING y funciones de agregado: Count(*), Count (columna), Avg( ), Sum( ), Min( ), Max( ) .....	103
c) Consultas basadas en más de una tabla .....	104
i) Producto cartesiano (FROM tabla1, tabla2).....	104
ii) Combinación común.....	104
(1) Con FROM t1, t2 WHERE t1.campoX = t2.campoY .....	104
(2) Con INNER JOIN.....	104
(a) LEFT .....	106
(b) RIGHT .....	106
(c) FULL .....	107
(d) Resumen de cuándo utilizar cada operación.....	107
iii) Combinación con alguna condición más (AND condición) .....	108
iv) Unión de filas (UNION).....	109
v) Diferencia (NOT EXISTS, NOT IN).....	110
vi) Intersección (EXISTS, INNER JOIN) .....	112
vii) División.....	113
d) Consulta de creación de una nueva tabla (SELECT INTO tabla) .....	114
e) Consulta en la creación de una vista (CREATE VIEW AS) .....	114
f) Subconsultas .....	115
i) Referencias externas .....	116
ii) Anidar subconsultas .....	117
8) Variables.....	119

a)Cast y Convert .....	121
9)Funciones .....	123
a)Funciones del sistema .....	123
b)Funciones definidas por el usuario .....	125
i)Funciones Escalares .....	125
ii)Funciones en línea .....	127
iii)Funciones de tabla de multisentencias .....	129
10)Procedimientos almacenados .....	132
11)Disparadores .....	137
a)Trigger DML. ....	137
b)Trigger DDL (Sólo para versiones superiores a la del 2005) .....	141
12)Flujos de control .....	142
a)La sentencia IF ... ELSE .....	142
b)La sentencia CASE .....	142
c)La sentencia WHILE .....	144

## 1) Bases de datos utilizadas para los ejemplos:

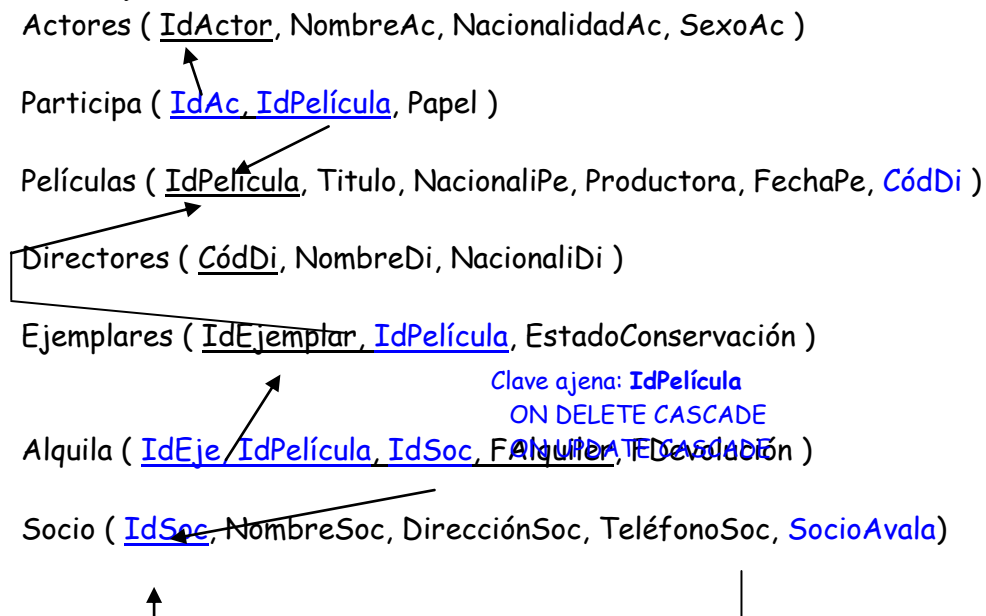
### a) VIDEOCLUBS GLOB-GUSTERS:

#### i) Esquema percibido:

La cadena de Videoclub Glob-Gusters ha decidido, para mejorar su servicio, emplear una base de datos para almacenar la información referente a las películas que ofrece en alquiler. Esta información es la siguiente:

1. Una película se caracteriza por su título, nacionalidad, productora y fecha (Por ejemplo, Quo Vadis, Estados Unidos, M.G.M., 1955).
2. En una película pueden participar varios actores (nombre, nacionalidad, sexo), algunos de ellos como actores principales.
3. Una película está dirigida por un director (nombre, nacionalidad).
4. De cada película se dispone de uno o varios ejemplares diferenciados por un número de ejemplar y caracterizados por su estado de conservación.
5. Un ejemplar se puede encontrar alquilado a algún cliente (nombre, dirección, teléfono). Se desea almacenar la fecha de comienzo del alquiler y la de devolución.
6. Cada socio puede alquilar como máximo 4 ejemplares.
7. Un socio tiene que ser avalado por otro socio que responda de él en caso de tener problemas en el alquiler.

#### ii) Grafo relacional:



**iii) Sentencias SQL para la creación:**

```

/*=====
VIDEOCLUB "GLOBGUSTERS"
=====
Creación de la Base de Datos
===== */

-- Pon en uso la base de datos Master
USE Master
GO

-- Comprobación, si existe la borra.
IF EXISTS (SELECT *      FROM master..sysdatabases
           WHERE name = N'VideoClub')
BEGIN
    -- Borra la Base de datos y la crea la base de datos.
    DROP DATABASE VideoClub
    CREATE DATABASE VideoClub
END
ELSE
BEGIN
    -- Crea la base de datos
    CREATE DATABASE VideoClub
END
GO

-- Pon en uso la base de datos
USE VideoClub
GO

----- Crea la tabla Actores
CREATE TABLE Actores(
    IdActor      smallint      IDENTITY (1, 1)
                  PRIMARY KEY,
    NombreAc     varchar (50)
    NacionaliAc  varchar (20)
    SexoAc       char (6) CHECK (SexoAc IN ('Hombre', 'Mujer'))
                  NOT NULL
)
GO

----- Crea la tabla Socios
CREATE TABLE Socios (
    IdSoc         smallint      IDENTITY (1, 1)
                  PRIMARY KEY,
    NombreSoc     varchar (50)
    DireccSoc     varchar (50)
    TeléfonoSoc   char (9) CHECK (TeléfonoSoc LIKE
    '9,6][0-9][0-9][0-9][0-9][0-9][0-9][0-9]',
    SocioAvala    smallint
                  NOT NULL
                  REFERENCES Socios (IdSoc)
)
GO

----- Crea la tabla Directores
CREATE TABLE Directores (
    CódDi         smallint      IDENTITY (1, 1)
                  PRIMARY KEY,
    NombreDi      varchar (50)
    NacionaliDi   varchar (20)
)
GO

```

```

----- Crea la tabla Películas
CREATE TABLE Películas (
    IdPelícula      smallint      IDENTITY (1, 1)
                        PRIMARY KEY,
    Título          varchar (50)      NOT NULL,
    NacionaliPe     varchar (20)      NOT NULL,
    Productora      varchar (30)      NOT NULL,
    FechaPe         smalldatetime     NOT NULL,
    CódDi           smallint          NOT NULL
                        REFERENCES Directores (CódDi)
)
GO

----- Crea la tabla Participa
CREATE TABLE Participa (
    Papel           varchar (20)      NOT NULL,
    IdActor         smallint          NOT NULL,
    IdPelícula      smallint          NOT NULL,
    CONSTRAINT PK_Participa PRIMARY KEY (IdActor, IdPelícula),
    CONSTRAINT FK_Participa1 FOREIGN KEY (IdActor)
                        REFERENCES Actores (IdActor),
    CONSTRAINT FK_Participa2 FOREIGN KEY (IdPelícula)
                        REFERENCES Películas (IdPelícula)
)
GO

----- Crea la tabla Ejemplares
CREATE TABLE Ejemplares (
    IdEjemplar      smallint      IDENTITY (1, 1) NOT NULL,
    IdPelícula      smallint      NOT NULL,
    EstadoCons      varchar (20)  NOT NULL,
    CONSTRAINT PK_Ejemplares PRIMARY KEY (IdEjemplar, IdPelícula),
    CONSTRAINT FK_Ejemplares FOREIGN KEY (IdPelícula)
                        REFERENCES Películas (IdPelícula)
                        ON DELETE CASCADE
                        ON UPDATE CASCADE
)
GO

----- Crea la tabla Alquila
CREATE TABLE Alquila (
    FechaAlqu       smalldatetime     NOT NULL,
    IdEjemplar      smallint          NOT NULL,
    IdPelícula      smallint          NOT NULL,
    IdSoc           smallint          NOT NULL,
    FechaDev        smalldatetime     NULL,
    CONSTRAINT PK_Alquila PRIMARY KEY (FechaAlqu, IdEjemplar, IdPelícula, IdSoc),
    CONSTRAINT FK_Alquila1 FOREIGN KEY (IdEjemplar, IdPelícula)
                        REFERENCES Ejemplares (IdEjemplar, IdPelícula),
    CONSTRAINT FK_Alquila2 FOREIGN KEY (IdSoc)
                        REFERENCES Socios (IdSoc),
)
GO

```



**iv) Sentencias SQL para la inserción:**

USE VideoClub

GO

----- Inserta registros en Actores

```

INSERT INTO Actores (NombreAc, NacionaliAc, SexoAc)
VALUES ('Antonio Banderas', 'Español', 'Hombre')
INSERT INTO Actores (NombreAc, NacionaliAc, SexoAc)
VALUES ('Patricia Conde', 'Española', 'Mujer')
INSERT INTO Actores (NombreAc, NacionaliAc, SexoAc)
VALUES ('Lucía la Piedra', 'Española', 'Mujer')
INSERT INTO Actores (NombreAc, NacionaliAc, SexoAc)
VALUES ('Pilar Rubio', 'Española', 'Mujer')
INSERT INTO Actores (NombreAc, NacionaliAc, SexoAc)
VALUES ('Brad Pitt', 'Americano', 'Hombre')
INSERT INTO Actores (NombreAc, NacionaliAc, SexoAc)
VALUES ('Robert de Niro', 'Americano', 'Hombre')
INSERT INTO Actores (NombreAc, NacionaliAc, SexoAc)
VALUES ('Jorge Javier Vazquez', 'Español', 'Hombre')

```

GO

----- Inserta registros en Socios

```

INSERT INTO Socios (NombreSoc, DireccSoc, TeléfonoSoc, SocioAvala)
VALUES ('Isaac Torralba', 'C/ Real, 3 - Santa Fe', '958741236', 1)
INSERT INTO Socios (NombreSoc, DireccSoc, TeléfonoSoc, SocioAvala)
VALUES ('Francisco Mariano Prieto', 'C/ La Calvario, 27 - Cozvíjar', '958781236', 1)
INSERT INTO Socios (NombreSoc, DireccSoc, TeléfonoSoc, SocioAvala)
VALUES ('Felipe Reyes', 'C/ Macarena, 43 - 5°C - Granada', '958223366', 2)
INSERT INTO Socios (NombreSoc, DireccSoc, TeléfonoSoc, SocioAvala)
VALUES ('David Rodríguez', 'C/ Picha Gorda - Granada', '693258741', 1)
INSERT INTO Socios (NombreSoc, DireccSoc, TeléfonoSoc, SocioAvala)
VALUES ('Francisco Barragán', 'Avd. Andalucía, 58 - Dúrcal', '632587419', 2)
INSERT INTO Socios (NombreSoc, DireccSoc, TeléfonoSoc, SocioAvala)
VALUES ('Antonio Megías', 'C/ Azucena, 1 - Santa Fe', '958741446', 3)
INSERT INTO Socios (NombreSoc, DireccSoc, TeléfonoSoc, SocioAvala)
VALUES ('Azucena Benítez', 'C/ RFederico García Lorca, 1 - Albuñuelas', '958776284', 2)
INSERT INTO Socios (NombreSoc, DireccSoc, TeléfonoSoc, SocioAvala)
VALUES ('ángela López', 'C/ Real, 113 - Otura', '958111111', 5)

```

GO

----- Inserta registros en Directores

```

INSERT INTO Directores (NombreDi, NacionaliDi)
VALUES ('Pedro Almodovar', 'Español')
INSERT INTO Directores (NombreDi, NacionaliDi)
VALUES ('Alejandro Amenabar', 'Español')
INSERT INTO Directores (NombreDi, NacionaliDi)
VALUES ('Antonio Banderas', 'Español')
INSERT INTO Directores (NombreDi, NacionaliDi)
VALUES ('Hermanos Cohen', 'Americano')
INSERT INTO Directores (NombreDi, NacionaliDi)
VALUES ('Mariano la Piedra', 'Polaco')

```

GO

```

----- Inserta registros en Películas
INSERT INTO Películas (Título, NacionaliPe, Productora, FechaPe, CódDi)
VALUES ('La bala que dobló la esquina', 'Española', 'Hermanos García S.A.', '23/10/1989', 5)
INSERT INTO Películas (Título, NacionaliPe, Productora, FechaPe, CódDi)
VALUES ('Los otros', 'Española', 'Producciones El aguila real', '12/12/1956', 2)
INSERT INTO Películas (Título, NacionaliPe, Productora, FechaPe, CódDi)
VALUES ('Mujeres al borde de un ataque de nervios', 'Española', 'Maricomix S.L.', '5/12/1963', 1)
INSERT INTO Películas (Título, NacionaliPe, Productora, FechaPe, CódDi)
VALUES ('CoCon', 'Americana', 'Columbia pictures', '8/5/1963', 5)
INSERT INTO Películas (Título, NacionaliPe, Productora, FechaPe, CódDi)
VALUES ('Ni quito ni pongo', 'Japonesa', 'Yujara', '15/11/1967', 5)
GO

----- Inserta registros en Participa
INSERT INTO Participa (Papel, IdActor, IdPelícula) VALUES ('Protagonista', 1, 1)
INSERT INTO Participa (Papel, IdActor, IdPelícula) VALUES ('Secundario', 1, 2)
INSERT INTO Participa (Papel, IdActor, IdPelícula) VALUES ('Protagonista', 4, 4)
INSERT INTO Participa (Papel, IdActor, IdPelícula) VALUES ('Secundario', 3, 5)
INSERT INTO Participa (Papel, IdActor, IdPelícula) VALUES ('Secundario', 1, 3)
INSERT INTO Participa (Papel, IdActor, IdPelícula) VALUES ('Protagonista', 2, 2)
GO

----- Inserta registros en Ejemplares
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (1, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (1, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (1, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (1, 'Regular')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (2, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (2, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (2, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (2, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (3, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (3, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (3, 'Regular')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (3, 'Regular')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (3, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (4, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (4, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (4, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (4, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (4, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (5, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (5, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (5, 'Bueno')
INSERT INTO Ejemplares (IdPelícula, EstadoCons) VALUES (5, 'Malo')
GO

```

```

----- Inserta registros en Alquila
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc)
VALUES ('24/11/2008', 1, 1, 1)
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc)
VALUES ('24/11/2008', 5, 2, 5)
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc, FechaDev)
VALUES ('24/11/2008', 13, 3, 4, '26/11/2008')
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc, FechaDev)
VALUES ('23/11/2008', 14, 4, 3, '27/11/2008')
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc, FechaDev)
VALUES ('23/11/2008', 2, 1, 3, '27/11/2008')
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc, FechaDev)
VALUES ('20/11/2008', 1, 1, 2, '20/11/2008')
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc, FechaDev)
VALUES ('21/11/2008', 7, 2, 6, '22/11/2008')
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc, FechaDev)
VALUES ('24/11/2008', 20, 5, 7, '26/11/2008')
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc, FechaDev)
VALUES ('24/11/2008', 4, 1, 7, '26/11/2008')
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc, FechaDev)
VALUES ('24/11/2008', 3, 1, 7, '26/11/2008')
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc, FechaDev)
VALUES ('12/11/2008', 2, 1, 8, '15/11/2008')
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc, FechaDev)
VALUES ('12/11/2008', 3, 1, 8, '15/11/2008')
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc, FechaDev)
VALUES ('27/11/2008', 3, 1, 1, '27/11/2008')
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc)
VALUES ('10/11/2008', 3, 1, 2)
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc)
VALUES ('10/11/2008', 17, 4, 2)
GO

```

**v) Sentencias SQL para la consulta de datos:**

```

/*=====
VIDEOCLUB
=====
Ejemplos sobre la Base de Datos
===== */

/* Pon en uso la base de datos VideoClub */
USE VideoClub
GO

/* -----
CONSULTAS
----- */

/* Mostrar el título y nombre del director de cada una de las películas que hay actualmente en la BD*/
SELECT Título, NombreDi
      FROM Películas, Directores
     WHERE Películas.CódDi = Directores.CódDi
-- De otra forma
SELECT Título, NombreDi
      FROM Películas
        INNER JOIN Directores
          ON Películas.CódDi = Directores.CódDi

/*¿Cuántas películas ha dirigido cada director?*/
SELECT NombreDi, Count(*) AS 'Número de películas'
      FROM Películas
        INNER JOIN Directores
          ON Películas.CódDi = Directores.CódDi
     GROUP BY Directores.NombreDi
-- De otra forma con alias para las tablas
SELECT NombreDi, Count(*) AS 'Número de películas'
      FROM Películas AS P
        INNER JOIN Directores AS D
          ON P.CódDi = D.CódDi
     GROUP BY D.NombreDi

/*¿Cuántas películas ha dirigido cada director que no sea Pedro Almodóvar?*/
SELECT NombreDi, Count(*) AS 'Número de películas'
      FROM Películas
        INNER JOIN Directores
          ON Películas.CódDi = Directores.CódDi
     WHERE NombreDi <> 'Pedro Almodovar'
     GROUP BY Directores.NombreDi

/*¿Cuántas películas ha dirigido cada director que no sea Pedro Almodóvar? ordenados de mayor número de películas a menor*/
SELECT NombreDi, Count(*) AS 'Número de películas'
      FROM Películas
        INNER JOIN Directores
          ON Películas.CódDi = Directores.CódDi
     WHERE NombreDi <> 'Pedro Almodovar'
     GROUP BY Directores.NombreDi
     ORDER BY 'Número de películas' DESC

```

```

/* Nombre de los actores, las películas y del director, de las películas dirigidas por Mariano la Piedara */
SELECT NombreAc, Título, NombreDi
  FROM Películas
    INNER JOIN Directores
      ON Películas.CódDi = Directores.CódDi
    INNER JOIN Participa
      ON Películas.IdPelícula = Participa.IdPelícula
    INNER JOIN Actores
      ON Participa.IdActor = Actores.IdActor
  WHERE NombreDi = 'Mariano la Piedara'

/* Nombre de las películas, el nombre del socio, y la fecha de alquiler de las películas que están alquiladas */
SELECT Título, NombreSoc, FechaAlqu
  FROM Películas
    INNER JOIN Ejemplares
      ON Películas.IdPelícula = Ejemplares.IdPelícula
    INNER JOIN Alquiler
      ON Ejemplares.IdEjemplar = Alquiler.IdEjemplar
    INNER JOIN Socios
      ON Alquiler.IdSoc = Socios.IdSoc
  WHERE FechaDev IS NULL

/* Crear una vista llamada 'películas_prestadas' donde se muestre Nombre de las películas, el nombre del socio, y la
fecha de alquiler de las películas que están alquiladas */
CREATE VIEW películas_prestadas AS (
  SELECT Título, NombreSoc, FechaAlqu
    FROM Películas
      INNER JOIN Ejemplares
        ON Películas.IdPelícula = Ejemplares.IdPelícula
      INNER JOIN Alquiler
        ON Ejemplares.IdEjemplar = Alquiler.IdEjemplar
      INNER JOIN Socios
        ON Alquiler.IdSoc = Socios.IdSoc
    WHERE FechaDev IS NULL
)

/* Mostrar las películas prestadas usando la vista anterior */
SELECT * FROM películas_prestadas

/* Mostrar los estados de conservación los ejemplares, indicando el estado y el número de ejemplares de cada estado de
conservación */
SELECT EstadoCons, Count(*) AS 'Número de ejemplares'
  FROM Ejemplares
  GROUP BY EstadoCons

/* Obtener los títulos de las películas que tengan más de 4 ejemplares */
SELECT Título, Count(*) AS 'Número de ejemplares'
  FROM Películas
    INNER JOIN Ejemplares
      ON Películas.IdPelícula = Ejemplares.IdPelícula
  GROUP BY Título
  HAVING Count(*) > 4

/* Indicar cuántas películas hay de cada nacionalidad */
SELECT NacionaliPe, Count(*) AS 'Número de ejemplares'
  FROM Películas
  GROUP BY NacionaliPe

```

```
/* ¿Cuántos actores hay de cada sexo? */
SELECT SexoAc, Count(*) AS 'Número de ejemplares'
FROM Actores
GROUP BY SexoAc

/* Muestra los distintos estados de conservación posibles de los ejemplares (sin repeticiones) */
SELECT DISTINCT EstadoCons
FROM Ejemplares

/* -----
TABLAS TEMPORALES
----- */

/* Crear una tabla temporal llamada 'Temporal1' sobre la tabla alquiler, con los datos IdEjemplar e IdPelícula para las películas que no estén devueltas. Dicha tabla temporal tendrá un carácter local.*/
SELECT IdEjemplar, IdPelícula INTO #Temporal1
FROM Alquiler
WHERE FechaDev IS NULL
-- Mostramos el contenido de la tabla temporal llamada 'temporal':
SELECT * FROM #Temporal1
-- Posteriormente la vamos a borrar:
DROP TABLE #Temporal1

/* Crear una tabla temporal llamada 'Temporal2' con los campos IdEjemplar e IdPelícula, ambos de tipo smallint, definiendo la como clave principal la unión de los dos. Dicha tabla temporal tendrá un carácter local.*/
CREATE TABLE #temporal2(
    IdEjemplar smallint,
    IdPelícula smallint,
    PRIMARY KEY (IdEjemplar, IdPelícula)
)
-- Insertamos un registro en la tabla con los valores 4 y 1.
INSERT INTO #Temporal2 VALUES (4,1)
-- Mostramos el contenido
SELECT * FROM #Temporal2
-- Posteriormente la vamos a borrar:
DROP TABLE #Temporal2
```

```

/* -----
INDICES
----- */
/* Creamos un índice sobre la tabla películas para el campo Título, comprobando que no exista, y si así fuese borrándolo.
*/
-- Se desactivan los mensajes de salida
SET NOCOUNT OFF
-- si existe el índice lo borra
IF EXISTS (
    SELECT name
    FROM sysindexes
    WHERE name = 'Ind_Título'
)
    DROP INDEX Películas.Título_ind -- El índice se menciona con tabla.nombreíndice
GO
-- Crea el índice
CREATE INDEX Ind_Título -- Creación de índice
    ON Películas (Título) -- Sobre Películas con el atributo Título
GO
-- Se activan los mensajes de salida
SET NOCOUNT ON

/* Creamos un índice sobre la tabla socios para el campo Título, comprobando que no exista, y si así fuese borrándolo.*/
-- si existe el índice lo borra
IF EXISTS (
    SELECT name
    FROM sysindexes
    WHERE name = 'Ind_NombreSoc'
)
    DROP INDEX Socios.Ind_NombreSoc
GO
-- Crea el índice
CREATE INDEX Ind_NombreSoc
    ON Socios(NombreSoc)
GO

/* Creamos un índice sobre la tabla directores para el campo NombreDi. */
CREATE INDEX Ind_NombreDi
    ON Directores (NombreDi)

/* Creamos un índice sobre la tabla actores para el campo NombreAc. */
CREATE INDEX NombreAc_ind
    ON Actores (NombreAc)

```

```
/* -----  
MIAS  
----- */  
  
/* Mostrar los títulos de las películas de nacionalidad 'Española' */  
SELECT Título, NacionaliPe  
FROM Películas  
WHERE NacionaliPe = 'Española'  
  
/* Mostrar los títulos de las películas cuya nacionalidad comience por 'A' */  
SELECT Título, NacionaliPe  
FROM Películas  
WHERE NacionaliPe LIKE ('A%')  
  
/* Por ejemplo para mostrar los títulos de las películas cuya nacionalidad no comience por 'A' */  
SELECT Título, NacionaliPe  
FROM Películas  
WHERE NacionaliPe NOT LIKE ('A%')  
  
/* Mostrar los Identificadores y títulos de las películas, cuyo identificador este entre 1 y 3 */  
SELECT IdPelícula, Título  
FROM Películas  
WHERE IdPelícula BETWEEN 1 AND 3  
  
/* Mostrar los Identificadores y títulos de las películas, cuyo identificador sea 1, 2 o 3 */  
SELECT IdPelícula, Título  
FROM Películas  
WHERE IdPelícula IN (1, 2, 3)  
  
/* Mostrar los códigos de las películas que no estén devueltas.*/  
SELECT IdPelícula  
FROM Alquiler  
WHERE FechaDev IS NULL  
  
/* Mostrar los códigos de las películas que estén devueltas.*/  
SELECT IdPelícula  
FROM Alquiler  
WHERE FechaDev IS NOT NULL  
  
/* Mostrar los títulos de las películas de nacionalidad 'Española' ordenados por el título */  
SELECT Título, NacionaliPe  
FROM Películas  
WHERE NacionaliPe = 'Española'  
ORDER BY Título  
  
/* Mostrar los títulos de las películas de nacionalidad 'Española' ordenados descendientemente por el título */  
SELECT Título, NacionaliPe  
FROM Películas  
WHERE NacionaliPe = 'Española'  
ORDER BY Título DESC  
  
/* Mostrar los títulos de las películas ordenados descendientemente por la nacionalidad y el ascendientemente por el título */  
SELECT Título, NacionaliPe  
FROM Películas  
ORDER BY NacionaliPe DESC, Título  
  
/* Mostrar la suma de los identificadores de las películas que tenemos */  
SELECT SUM(IdPelícula)  
FROM Películas
```



```
71/* Mostrar el número de películas que tenemos */
SELECT Count(*)
    FROM Películas

/* Mostrar la media de los identificadores de las películas que tenemos: */
SELECT Avg(IdPelícula)
    FROM Películas

/* Mostrar el identificador mas grande que tenemos: */
SELECT Max(IdPelícula)
    FROM Películas

/* Mostrar el identificador mas pequeño que tenemos: */
SELECT Min(IdPelícula)
    FROM Películas

/*Mostrar los estados de conservación de los ejemplares, indicando el estado y el número de ejemplares de cada estado
de conservación, pero solamente de los que tenga más de 5 ejemplares.*/
SELECT EstadoCons, Count(*) AS 'Número de ejemplares'
    FROM Ejemplares
    GROUP BY EstadoCons
    HAVING Count(*)>5

/* Mostrar la suma de los identificadores de las películas que tenemos:*/
SELECT Sum(IdPelícula) AS 'Suma IdPelícula'
    FROM Películas

/* Mostrar el número de películas que tenemos: */
SELECT Count(*) AS 'Nº Películas'
    FROM Películas

/* Mostrar la media de los identificadores de las películas que tenemos:*/
SELECT Avg(IdPelícula) AS 'Media IdPelícula'
    FROM Películas

/* Mostrar el identificador más grande que tenemos:*/
SELECT Max(IdPelícula) AS 'IdPelícula Mayor'
    FROM Películas

/* Mostrar el identificador más pequeño que tenemos: */
SELECT Min(IdPelícula) AS 'IdPelícula Menor'
    FROM Películas

/* Mostrar todas las filas de la tabla películas.*/
SELECT *
    FROM Películas

/* Mostrar los Identificadores y títulos de las películas, cuyo identificador sea 1, 2 o 3 */
SELECT IdPelícula, Título
    FROM Películas
    WHERE IdPelícula IN (1, 2, 3)

/* Mostrar los identificadores y títulos de las películas, que estén alquiladas y no devueltas. */
SELECT IdPelícula, Título
    FROM Películas
    WHERE IdPelícula IN (SELECT DISTINCT IdPelícula
                        FROM Alquila
                        WHERE FechaDev IS NULL)
```

```
/* Mostrar los Ejemplares que han sido alquilados. (Intersección)*/
SELECT IdEjemplar
  FROM Ejemplares
 WHERE IdEjemplar IN (
      SELECT DISTINCT IdEjemplar
      FROM Alquila
    )

/* Mostrar las fechas en que han alquilado algún ejemplar y se ha devuelto alguno también. (Intersección)*/
SELECT FechaAlqu
  FROM Alquila
 WHERE FechaAlqu IN (
      SELECT FechaDev
      FROM Alquila
    )

/* Mostrar el código de los directores que han dirigido alguna película. (Intersección)*/
SELECT CódDi
  FROM Directores
 WHERE CódDi = ANY(
      SELECT DISTINCT CódDi
      FROM Películas
    )

/* Mostrar las nacionalidades de las que hay algún actor y hay algún director. (Intersección)*/
SELECT DISTINCT NacionaliAC
  FROM Actores
 WHERE NacionaliAc IN (
      SELECT NacionaliDi
      FROM Directores
    )

/* Devuelve los socios que han realizado algún alquiler. (intersección)*/
SELECT NombreSoc
  FROM Socios
 WHERE EXISTS (
      SELECT *
      FROM Alquila
      WHERE Socios.IdSoc = Alquila.IdSoc
    )

/* Mostrar los ejemplares que no han sido alquilados. (Diferencia)*/
SELECT IdEjemplar
  FROM Ejemplares
 WHERE IdEjemplar NOT IN (
      SELECT DISTINCT IdEjemplar
      FROM Alquila
    )

/* Mostrar las fechas en que se ha alquilado algún ejemplar y no se ha devuelto ninguno. (Diferencia)*/
SELECT FechaAlqu
  FROM Alquila
 WHERE FechaAlqu NOT IN (
      SELECT FechaAlqu
      FROM Alquila
      WHERE FechaDev IS NULL
    )
```

```

/* Mostrar los códigos de los directores que no han dirigido ninguna película. (Diferencia)*/
SELECT CódDi
  FROM Directores
 WHERE NOT EXISTS (
      SELECT CódDi
      FROM Películas
      WHERE Directores.CódDi = CódDi
    )
--O bien:
SELECT CódDi
  FROM Directores
 WHERE CódDi <> ALL (
      SELECT DISTINCT CódDi
      FROM Películas
    )

/* Mostrar las nacionalidades de las que hay algún actor y no hay algún director. (Diferencia)*/
SELECT DISTINCT NacionaliAc
  FROM Actores
 WHERE NacionaliAc NOT IN (
      SELECT NacionaliDi
      FROM Directores
    )

/* Mostrar las Nacionalidades de los actores que no estén en directores. (Diferencia)*/
SELECT DISTINCT NacionaliDi
  FROM Directores
 WHERE NacionaliDi NOT IN (
      SELECT DISTINCT NacionaliAc
      FROM Actores
    )

/* Mostrar todas las nacionalidades de las que hay algún actor o algún director. (Unión)*/
SELECT NacionaliAc
  FROM Actores
UNION
SELECT NacionaliDi
  FROM Directores

/* Mostrar una tabla con los nombres de los Actores y de los directores, además le añadimos el tipo según sea un actor o
director (Unión).*/
SELECT 'Tipo' = 'Actor', NombreAc
  FROM Actores
UNION
SELECT 'Tipo' = 'Director', NombreDi
  FROM Directores

```

```

/* Saber el actor o los actores que han participado en las tres películas especificadas. (División) */
-- Para hacer el divisor
CREATE VIEW Películas2 AS (
    SELECT DISTINCT Películas.IdPelícula
    FROM Películas, Participa
    WHERE Películas.IdPelícula = Participa.IdPelícula AND
        Título IN ( 'La bala que dobló la esquina', 'Los otros',
            'Mujeres al borde de un ataque de nervios')
)
GO
-- Saca el resultado
SELECT DISTINCT IdActor, NombreAc
    FROM Actores
    WHERE NOT EXISTS (
        SELECT *
            FROM Películas2
            WHERE NOT EXISTS (
                SELECT *
                    FROM Participa
                    WHERE Participa.IdActor = Actores.IdActor And
                        Participa.IdPelícula = Películas2.IdPelícula
            )
        )
)

/* Mostrar todas las filas donde la fecha de alquiler de la tabla Alquiler esté entre el 01/06/2008 y el 31/12/2008 y no
están devueltas*/
SELECT *
    FROM Alquiler
    WHERE FechaAlqu BETWEEN '01/06/2008' AND '31/12/2008' AND
        FechaDev IS NULL

/* Para mostrar los títulos de las películas que empiecen por 'C'.*/
SELECT Título
    FROM Películas
    WHERE Título LIKE ('C%')

/* Mostrar los Directores que hayan dirigido alguna película */
SELECT NombreDi
    FROM Directores
    WHERE CódDi = ANY (SELECT CódDi FROM Películas)

/*Mostrar todos Directores que no hayan dirigido.*/
SELECT CódDi
    FROM Directores
    WHERE CódDi <> ALL (SELECT DISTINCT CódDi FROM Películas)

/* Muestra los títulos de las películas que tengan al menos un ejemplar en la tabla de ejemplares.*/
SELECT Título
    FROM Películas
    WHERE EXISTS (SELECT * FROM Ejemplares
        WHERE Películas.IdPelícula = Ejemplares.IdPelícula)

/* Muestra los títulos de las películas que NO tengan al menos un ejemplar en la tabla de ejemplares.*/
SELECT Título
    FROM Películas
    WHERE NOT EXISTS (SELECT * FROM Ejemplares
        WHERE Películas.IdPelícula = Ejemplares.IdPelícula)

/* Muestra los Códigos de todas las películas que se hallan alquilado, aunque estén repetidas..*/
SELECT ALL IdPelícula
    FROM Alquiler

```

```
/* Las siguientes instrucciones mostrarían los identificadores de las películas alquiladas de forma única */
SELECT DISTINCT IdPelícula
FROM Alquila

/* Muestra los códigos de las películas que estén alquiladas pero sin que se repita el código de la película. */
SELECT DISTINCT IdPelícula
FROM Alquila
WHERE FechaDev IS NULL

/* Por ejemplo para mostrar el título de la película y la nacionalidad, utilizando alias para ambos campos. */
SELECT Título AS 'Título de la Película', NacionaliPE AS Nacionalidad
FROM Películas
-- O bien
SELECT Título AS 'Título de la Película', NacionaliPE AS Nacionalidad
FROM Películas AS P
-- O bien
SELECT Título 'Título de la Película', NacionaliPE Nacionalidad
FROM Películas P

/* Muestra el identificador de cada película alquilada, el número de días que estuvo alquilada, y el precio de cada alquiler sabiendo que se paga 0,5€ por día de alquiler. */
SELECT IdPelícula, DATEDIFF(dd, FechaAlqu, FechaDev) AS Días,
(DATEDIFF(dd, FechaAlqu, FechaDev ) * 0.5) AS Importe
FROM Alquila
WHERE FechaDev IS NOT NULL

/* Muestra cuantos días ha estado una película alquilada */
SELECT IdPelícula, DATEDIFF(dd, FechaAlqu, FechaDev) AS Días
FROM Alquila
WHERE FechaDev IS NOT NULL

/* Muestra los nombres de los tres primeros socios. */
SELECT TOP 3 NombreSoc
FROM Socios

/* Muestra los nombres del 25% del número total de socios. */
SELECT TOP 25 PERCENT NombreSoc
FROM Socios

/* Muestra los nombres del 10% de los primeros socios después de ordenarlos por orden descendente del nombre. */
SELECT TOP 30 PERCENT NombreSoc
FROM Socios
ORDER BY NombreSoc DESC

/* Muestra los nombres de los tres primeros socios después de ordenarlos por orden descendente del nombre. */
SELECT TOP 3 NombreSoc
FROM Socios
ORDER BY NombreSoc DESC

/* Muestra los nombres del 30% de los primeros socios después de ordenarlos por orden descendente del nombre. */
SELECT TOP 30 PERCENT NombreSoc
FROM Socios
ORDER BY NombreSoc DESC

/* Muestra el código del socio con los alquileres de películas no devueltas poniendo primero los alquileres más antiguos. */
SELECT IdSoc, FechaAlqu
FROM Alquila
WHERE FechaDev IS NULL
ORDER BY FechaAlqu DESC
```

```
/* Mostrar cuántas películas hay de cada nacionalidad.*/  
SELECT NacionaliPe, Count(*) AS 'Nº de ejemplares'  
FROM Películas  
GROUP BY NacionaliPe  
  
/* Muestra las películas hay de cada nacionalidad siempre que sean mayores o iguales que 3.*/  
SELECT NacionaliPe, Count(*) AS 'Nº de ejemplares'  
FROM Películas  
GROUP BY NacionaliPe  
HAVING Count(*)>=3  
  
/* Muestra el número de películas que tiene la tabla de películas.*/  
SELECT Count(*) AS 'Nº de películas'  
FROM Películas  
  
/* Muestra cuantos títulos distintos hay en la tabla de películas.*/  
SELECT Count(Título) AS 'Títulos distintos'  
FROM Películas  
  
/* Devuelve el producto Cartesiano de la tabla empleado y trabaja.*/  
SELECT *  
FROM Películas, Directores  
  
/* Para devolver todos nombres de actores y las películas que protagonizó aunque no tengan ninguna película, a estas se les pone el valor NULL. */  
SELECT NombreAc, Título  
FROM Actores  
LEFT JOIN Participa ON Participa.IdActor = Actores.IdActor  
LEFT JOIN Películas ON Películas.IdPelícula = Participa.IdPelícula  
  
/* Para devolver todos los nombres de directores y las películas que dirigió aunque no tenga ninguna película, a estas se les pone el valor NULL */  
SELECT Título, NombreDi  
FROM Películas  
RIGHT JOIN Directores ON Directores.CódDi = Películas.CódDi  
  
/* Para devolver todas las filas de las dos tabla, tanto las comunes como las no comunes */  
SELECT FechaDev, Título  
FROM Alquiler  
FULL JOIN Películas ON Películas.IdPelícula = Alquiler.IdPelícula  
  
/* Para devolver el nombre del actor y las películas en las que participa y que además el actor sea 'Antonio Banderas'. */  
SELECT NombreAc, Título  
FROM Actores  
INNER JOIN Participa  
ON Participa.IdActor = Actores.IdActor  
INNER JOIN Películas  
ON Películas.IdPelícula = Participa.IdPelícula AND Actores.NombreAc = 'Antonio Banderas'
```

```

/* -----
SUBCONSULTAS
----- */

/* Para consultar el nombre del socio que realizó el último alquiler */
SELECT NombreSoc
      FROM Socios
      WHERE IdSoc IN (
            SELECT TOP 1 IdSoc
            FROM Alquiler
            ORDER BY FechaAlqu DESC
        )

/* Para mostrar el nombre la fecha del último alquiler y la dirección de todos los socios. */
SELECT TOP 1 NombreSoc, (
        SELECT Max(FechaAlqu)
        FROM Alquiler
        WHERE IdSoc = Socios.IdSoc) AS 'Fecha último alquiler',
        DireccSoc
      FROM Socios

/* Para mostrar el título y la productora de las películas alquiladas y devueltas. */
SELECT Título, Productora
      FROM Películas
      WHERE IdPelícula IN (
            SELECT IdPelícula
            FROM Ejemplares
            WHERE IdEjemplar IN (
                  SELECT IdEjemplar
                  FROM Alquiler
                  WHERE FechaDev IS NULL
            )
        )

/* Para mostrar el título, la productora de los ejemplares, haríamos: */
SELECT Título, Productora
      FROM Películas
      WHERE IdPelícula IN (
            SELECT IdPelícula
            FROM Ejemplares
            WHERE Películas.IdPelícula = Ejemplares.IdPelícula
        )

-- Pero podría haberse escrito como:
SELECT DISTINCT Título, Productora
      FROM Películas, Ejemplares
      WHERE Películas.IdPelícula = Ejemplares.IdPelícula

-- También se podría haber escrito como:
SELECT DISTINCT Título
      FROM Películas
            INNER JOIN Ejemplares
            ON Películas.IdPelícula = Ejemplares.IdPelícula

/* Crea una vista sobre la tabla películas, en la que se nos muestren los distintos títulos que tenemos en nuestro
VideoClub, se haría de la siguiente forma. */
CREATE VIEW DistTítulos AS (
        SELECT DISTINCT Título
        FROM Películas
    )
SELECT * FROM DistTítulos

```

```

/* -----
PROCEDIMIENTOS ALMACENADOS
----- */

/* Contar el número de ejemplares de una película*/
CREATE PROCEDURE pa_CuentaPelículas
    @Buscado varchar(50),
    @Número smallint OUTPUT
AS
    SELECT @Número = Count(*)
        FROM Películas
            INNER JOIN Ejemplares
                ON Películas.IdPelícula = Ejemplares.IdPelícula
            WHERE Películas.Título = @Buscado

GO
-- Comprobación de su funcionamiento
DECLARE @NúmeroPel int
SET @NúmeroPel=0
EXEC pa_CuentaPelículas 'CoCon', @NúmeroPel OUTPUT
PRINT @NúmeroPel
--Borra el procedimiento almacenado
DROP PROCEDURE pa_CuentaPelículas

/* Pasarle un código de una película y un nuevo estado de conservación. Si existe que lo cambie, y si no que nos diga que no
existe.*/
CREATE PROCEDURE pa_CambiaEstadoConservación
    @IdEjemplar smallint,
    @NuevoEstCons varchar(50)
AS
    DECLARE @Salida smallint
    SET @Salida = 1
    IF EXISTS (SELECT *
        FROM Ejemplares
        WHERE IdEjemplar = @IdEjemplar)
        BEGIN
            UPDATE Ejemplares
                SET EstadoCons = @NuevoEstCons
                WHERE IdEjemplar = @IdEjemplar
            RETURN 0      -- Salida exitosa
        END
    ELSE
        BEGIN
            PRINT 'Ejemplar no coincidente. Error: 1'
            RETURN 1      -- Salida fallida
        END
    END

GO
-- Comprobación de su funcionamiento
SELECT *
    FROM Ejemplares
    WHERE IdEjemplar = 1
DECLARE @RP int
EXEC @RP = pa_CambiaEstadoConservación 1, 'Malísimo'
PRINT @RP
SELECT *
    FROM Ejemplares
    WHERE IdEjemplar = 1
--Borra el procedimiento almacenado
DROP PROCEDURE pa_CambiaEstadoConservación

```



```
/* Procedimiento que se le pase una Fecha de alquiler y devuelva Nombre del Socio y Título de la películas alquiladas en esa fecha.*/
CREATE PROCEDURE pa_InfoFecha
    @Fecha smalldatetime
AS
    IF EXISTS (SELECT *
                FROM Alquila
                WHERE FechaAlqu = @Fecha)
        BEGIN
            SELECT NombreSoc, Título
            FROM Alquila
            INNER JOIN Socios
            ON Alquila.IdSoc = Socios.IdSoc
            INNER JOIN Películas
            ON Alquila.IdPelícula = Películas.IdPelícula
            WHERE FechaAlqu = @Fecha And FechaDev IS NULL
            RETURN 0      -- Salida exitosa
        END
    ELSE
        BEGIN
            PRINT 'Fecha sin alquileres. Error: 1'
            RETURN 1      -- Salida fallida
        END
GO
-- Comprobación de su funcionamiento
SELECT *
    FROM ALQUILA

DECLARE @RP int
EXEC @RP = pa_InfoFecha '12/11/2008'
PRINT @RP
--Borra el procedimiento almacenado
DROP PROCEDURE pa_InfoFecha
```

```

/* -----
DISPARADORES
----- */
-- Crea la tabla ControlAlquileres
CREATE TABLE ControlAlquileres (
    NúmeroControl    smallint          IDENTITY (1, 1)
                                PRIMARY KEY,
    Título            varchar (50)
    IdEjemplar        smallint          NOT NULL,
    FechaAlquiler     smalldatetime    DEFAULT GetDate()
)
GO
-- Crea el disparador
CREATE TRIGGER tr_AñadeHistório
    ON Alquila
    FOR INSERT
AS
BEGIN
    -- Inserto los datos en la tabla nueva
    INSERT INTO ControlAlquileres (Título, IdEjemplar)
        SELECT Películas.Título, IdEjemplar
            FROM Inserted
                INNER JOIN Películas
                    ON Inserted.IdPelícula = Películas.IdPelícula
END
-- Para probar el funcionamiento del disparador
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc)
    VALUES (GetDate(), 2, 1, 1)
SELECT *
    FROM ControlAlquileres
-- Desactiva el disparador tr_AñadeHistório de la tabla Alquila
ALTER TABLE Alquila DISABLE TRIGGER tr_AñadeHistório
GO
-- Activa el disparador tr_AñadeHistório de la tabla alquila
ALTER TABLE Alquila ENABLE TRIGGER tr_AñadeHistório
GO
-- Desactiva todos los disparadores de la tabla Alquila
ALTER TABLE Alquila DISABLE TRIGGER ALL
GO
-- Activa todos los disparadores de la tabla Alquila
ALTER TABLE Alquila ENABLE TRIGGER ALL
GO
-- Borra el disparador
DROP TRIGGER tr_AñadeHistório
GO

```

```

/* Crear un disparador para comprobar que un Socio no pueda alquilar más de 3 ejemplares.*/
CREATE TRIGGER tr_MaximoAlquileresSocio
    ON Alquila
    FOR INSERT
AS
BEGIN
    -- Calculo el número de Alquileres del socio
    DECLARE @NúmeroAlquileres smallint
    SELECT @NúmeroAlquileres = Count (*)
        FROM Alquila
        WHERE FechaDev IS NULL And
            IdSoc = (SELECT IdSoc
                    FROM Inserted)
    -- Compruebo el número de Alquileres del socio
    IF @NúmeroAlquileres > 3
    BEGIN
        PRINT 'Ha superado el número máximo de alquileres permitido (3)'
        ROLLBACK
    END
END
-- Para probar el funcionamiento del disparador
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc)
    VALUES (GetDate(), 2, 1, 2)
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc)
    VALUES (GetDate(), 3, 1, 2)
-- Borra el disparador
DROP TRIGGER tr_MaximoAlquileresSocio
GO

/* Borre las filas de la tabla ControlAlquileres cuando el cliente devuelva una película */
CREATE TRIGGER tr_DevuelvePelícula
    ON Alquila
    AFTER UPDATE
AS
BEGIN
    IF UPDATE (FechaDev)      -- Sólo si se actualiza la fecha de devolución
    BEGIN
        -- Inicializa la variable necesaria
        DECLARE @IdEjemplar smallint
        SELECT @IdEjemplar = IdEjemplar
            FROM Inserted
        -- Borro la línea de ControlAlquileres
        DELETE
            FROM ControlAlquileres
            WHERE IdEjemplar = @IdEjemplar
    END
END
-- Para probar el funcionamiento del disparador
UPDATE Alquila
    SET FechaDev = GetDate()
    WHERE IdSoc = 1 And
        IdPelícula = 1 And
        IdEjemplar = 3

```

```

/* -----
FUNCIONES
----- */
/* Función a la que pasamos un título de una película y devuelve el valor del nombre del director de la película */
CREATE FUNCTION DirectorPelícula ( @Título varchar(50) )
    RETURNS          varchar(50)
AS
BEGIN
    DECLARE @Director varchar(50)
    SELECT @Director = NombreDi
        FROM Películas
            INNER JOIN Directores
                ON Películas.CódDi = Directores.CódDi
        WHERE Título = @Título
    RETURN @Director
END
-- Llamada a la función anterior sería el siguiente
PRINT dbo.DirectorPelícula('Cocon')
-- Borra la función creada
DROP FUNCTION DirectorPelícula

/* Función a la que pasamos un título de una película y devuelve el número de años en que se rodó la película */
CREATE FUNCTION AñosPelícula ( @Título varchar(50), @FechaActual smalldatetime )
    RETURNS          varchar(50)
AS
BEGIN
    DECLARE @Años smallint, @Hoy smallint
    --SELECT @Años = Year(@FechaActual) - Year(FechaPe)
    -- O bien:
    SELECT @Años = DATEDIFF(year, FechaPe, @FechaActual)
        FROM Películas
        WHERE Título = @Título
    RETURN @Años
END
-- Llamada a la función anterior sería el siguiente
PRINT dbo.AñosPelícula('Cocon', GetDate())
-- Borra la función creada
DROP FUNCTION AñosPelícula

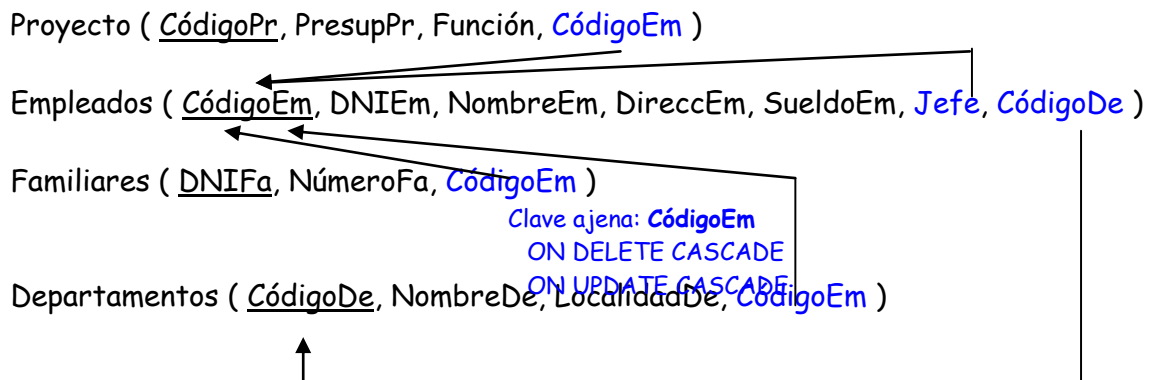
```

```
/* -----  
FLUJOS DE CONTROL  
----- */  
  
-- EJEMPLOS CASE -----  
  
-- Ejemplo 1, un campo al que le añadimos ' (ES)' y otra columna que indica si es Actor o Actriz  
SELECT NombreAc + ' (ES)' AS 'Aañadido', Tipo = CASE SexoAc  
                WHEN 'Hombre' THEN 'Actor'  
                WHEN 'Mujer'   THEN 'Actriz'  
                END  
FROM Actores  
WHERE NacionaliAc LIKE ('Españ%')
```

**b) PROYECTALIA:****i) Esquema percibido:**

Se trata de realizar el esquema E/R de la empresa PROYECTALIA, dedicada a la realización de proyectos, que está organizada por departamentos y que tiene una serie de empleados, algunos de los cuales tienen a su cargo (como jefe) varios empleados.

1. Un empleado no puede tener más que un jefe.
2. Cada empleado pertenece a un departamento y sólo a uno.
3. Cada departamento tiene un único empleado como responsable. Los empleados que no son responsables de un departamento trabajan en uno o varios proyectos y cada proyecto puede tener trabajando en él a ninguno o a varios empleados.
4. De cara a la paga para ayuda familiar, el sistema deberá reflejar los familiares de cada empleado (el que los tenga).
5. De los empleados, interesa almacenar el Cód\_Emp (Atributo Identificador Primario), el DNI, nombre, dirección y sueldo.
6. De cada departamento interesa el Cód\_Dep (AIP), nombre y localidad.
7. De los proyectos, registraremos el Cód\_Proj (AIP) y el presupuesto.
8. De los familiares, el DNI (AIP) y el nombre.
9. Interesa, además, recoger la función que un empleado desempeña en el proyecto en que trabaja.

**ii) Grafo relacional:**

**iii) Sentencias SQL para la creación:**

```

/*=====
PROYECTALIA
=====
Creación de la Base de Datos
===== */

-- Pon en uso la base de datos Master
USE Master
GO

-- Comprobación, si existe la borra.
IF EXISTS (SELECT * FROM master..sysdatabases
           WHERE name = N'Proyectalia')
BEGIN
    -- Borra la Base de datos y la crea la base de datos.
    DROP DATABASE Proyectalia
    CREATE DATABASE Proyectalia
END
ELSE
BEGIN
    -- Crea la base de datos
    CREATE DATABASE Proyectalia
END
GO

-- Pon en uso la base de datos
USE Proyectalia
GO

----- Crea la tabla Proyecto
CREATE TABLE Proyecto (
    CódigoPr          int          IDENTITY (1, 1)
                        PRIMARY KEY,
    PresupPr          smallmoney   NOT NULL,
    Función            varchar (50) NOT NULL,
    CódigoEm          smallint     NULL
    -- Necesita modificaciones (al final).
    -- Clave ajena (CódigoEm)
)
GO

----- Crea la tabla Empleados
CREATE TABLE Empleados (
    CódigoEm          smallint     IDENTITY (1, 1)
                        PRIMARY KEY,
    DNIEm             char (9) CHECK (DNIEm LIKE
    '[0-9A-Z][0-9][0-9][0-9][0-9][0-9][0-9][0-9A-Z]')
                        UNIQUE      NOT NULL,
    NombreEm          varchar (50) NOT NULL,
    DireccEm          varchar (50) NOT NULL,
    SueldoEm          smallmoney   NOT NULL,
    Jefe              smallint     NULL,
    CódigoDe          smallint     DEFAULT 1
    -- Necesita modificaciones (al final).
    -- Clave ajena (Jefe, CódigoDe)
)
GO

```

```

----- Crea la tabla Familiares
CREATE TABLE Familiares (
    DNIFa          char (9) CHECK (DNIFa LIKE
                        '[0-9A-Z][0-9][0-9][0-9][0-9][0-9][0-9][0-9A-Z]')
                        PRIMARY KEY,
    NúmeroFa      smallint          DEFAULT 1,
    CódigoEm      smallint          REFERENCES Empleados (CódigoEm)
                        ON DELETE CASCADE
                        ON UPDATE CASCADE
)
GO

----- Crea la tabla Departamentos
CREATE TABLE Departamentos (
    CódigoDe      smallint          IDENTITY (1, 1)
                        PRIMARY KEY,
    NombreDe      varchar (50)      NOT NULL,
    LocalidadDe   varchar (50)      DEFAULT 'Granada',
    CódigoEm      smallint          NULL
    -- Necesita modificaciones (al final).
    -- Clave ajena (CódigoEm)
)
GO

-----
-- MODIFICACIONES
----- Modifica la tabla Proyecto
    -- Clave ajena (CódigoEm)
ALTER TABLE Proyecto ADD
    CONSTRAINT FK_CódEmPro      FOREIGN KEY (CódigoEm) REFERENCES Empleados(CódigoEm)

----- Modifica la tabla Empleados
    -- Clave ajena (Jefe, CódigoDe)
    -- Debemos insertar el primer Empleado, y el primer Departamento tiene que estar creado
ALTER TABLE Empleados ADD
    CONSTRAINT FK_Jefe          FOREIGN KEY (Jefe)      REFERENCES Empleados(CódigoEm),
    CONSTRAINT FK_CódigoDe     FOREIGN KEY (CódigoDe) REFERENCES Departamentos(CódigoDe)

----- Modifica la tabla Departamentos
    -- Clave ajena (CódigoEm)
ALTER TABLE Departamentos ADD
    CONSTRAINT FK_CódEmDep     FOREIGN KEY (CódigoEm) REFERENCES Empleados (CódigoEm)

```



**iv) Sentencias SQL para la inserción de datos:**

```

/*=====
EL RÁPIDO
=====
Inserción en la Base de Datos
===== */

-- Pon en uso la base de datos
USE Proyectalia
GO

----- Inserta registros en Departamentos
-- DESACTIVA la restricción del Código de empleado
ALTER TABLE Departamentos
      NOCHECK      CONSTRAINT      FK_CódEmDep
-- Inserto los registros
INSERT INTO  Departamentos (NombreDe, LocalidadDe, CódigoEm)  VALUES ('Administración', 'Cozviyar', 1)
INSERT INTO  Departamentos (NombreDe, LocalidadDe, CódigoEm)  VALUES ('Contabilidad', 'Santa Fé', 2)
INSERT INTO  Departamentos (NombreDe, CódigoEm)                VALUES ('Personal', 3)
-- ACTIVA la restricción del Código de empleado
ALTER TABLE Departamentos
      CHECK        CONSTRAINT      FK_CódEmDep
GO

----- Inserta registros en Empleados
INSERT INTO  Empleados (DNIEm, NombreEm, DireccEm, SueldoEm, CódigoDe)
VALUES ('11111111A', 'Nombre del empleado 1', 'La casa del empleado 1', €100.0, 1)
INSERT INTO  Empleados (DNIEm, NombreEm, DireccEm, SueldoEm, Jefe, CódigoDe)
VALUES ('22222222B', 'Nombre del empleado 2', 'La casa del empleado 2', €200.0, 1, 2)
INSERT INTO  Empleados (DNIEm, NombreEm, DireccEm, SueldoEm, Jefe, CódigoDe)
VALUES ('33333333C', 'Nombre del empleado 3', 'La casa del empleado 3', €300.0, 2, 3)
INSERT INTO  Empleados (DNIEm, NombreEm, DireccEm, SueldoEm, Jefe, CódigoDe)
VALUES ('44444444D', 'Nombre del empleado 4', 'La casa del empleado 4', €400.0, 3, 1)
INSERT INTO  Empleados (DNIEm, NombreEm, DireccEm, SueldoEm, Jefe, CódigoDe)
VALUES ('55555555E', 'Nombre del empleado 5', 'La casa del empleado 5', €500.0, 4, 2)
GO

----- Inserta registros en Proyecto
INSERT INTO  Proyecto (PresupPr, Función, CódigoEm)  VALUES (€100, 'Proyecto número 1', 1)
INSERT INTO  Proyecto (PresupPr, Función, CódigoEm)  VALUES (€200, 'Proyecto número 2', 2)
INSERT INTO  Proyecto (PresupPr, Función, CódigoEm)  VALUES (€300, 'Proyecto número 3', 3)
INSERT INTO  Proyecto (PresupPr, Función, CódigoEm)  VALUES (€400, 'Proyecto número 4', 4)
INSERT INTO  Proyecto (PresupPr, Función, CódigoEm)  VALUES (€500, 'Proyecto número 5', 1)
GO

----- Inserta registros en Familiares
INSERT INTO  Familiares (DNIFa, NúmeroFa, CódigoEm)  VALUES ('10000000A', 3, 1)
INSERT INTO  Familiares (DNIFa, CódigoEm)            VALUES ('20050000B', 2)
INSERT INTO  Familiares (DNIFa, CódigoEm)            VALUES ('30000000C', 3)
INSERT INTO  Familiares (DNIFa, CódigoEm)            VALUES ('40000000D', 4)
INSERT INTO  Familiares (DNIFa, NúmeroFa, CódigoEm)  VALUES ('50000000E', 2, 5)
GO

```

**v) Sentencias SQL para la consulta de datos:**

```

/*=====
PROYECTALIA
=====
Consultas sobre la Base de Datos
===== */

-- Pon en uso la base de datos
USE Proyectalia
GO

/* Por ejemplo, creamos un índice sobre la columna DNIEm en la tabla Empleados una vez creada la tabla */
CREATE UNIQUE INDEX Ind_DNIEm ON Empleados (DNIEm)

/* Por ejemplo deberemos introducir estas sentencias para añadir un campo llamado UbicaciónDpto a la tabla
Departamentos de la base de datos Proyectalia: */
ALTER TABLE Departamentos
    ADD UbicaciónDpto smallint

/* Eliminar el campo anteriormente creado. */
ALTER TABLE Departamentos
    DROP COLUMN UbicaciónDpto

/* Para modificar el tipo de datos almacenado en el campo UbicaciónDpto de la tabla Departamentos de la base de datos
Proyectalia deberemos introducir las siguientes sentencias: */
ALTER TABLE Departamentos
    ALTER COLUMN UbicaciónDpto varchar(10)

/* Por ejemplo para borrar de la base de datos Proyectalia, la columna UbicaciónDpto de la tabla Departamentos
deberemos introducir estas sentencias: */
ALTER TABLE Departamentos
    DROP COLUMN UbicaciónDpto

/* Para deshabilitar la restricción de clave foránea que tiene la tabla de Departamentos tendríamos que hacer lo
siguiente:*/
ALTER TABLE Departamentos
    NOCHECK CONSTRAINT FK_CódEmDep

/* Para habilitar una restricción deshabilitada, se ejecuta la misma instrucción pero con la cláusula "CHECK" o "CHECK ALL",
sobre el ejemplo anterior tendríamos que:*/
ALTER TABLE Departamentos
    CHECK CONSTRAINT FK_CódEmDep

/* Para borrar la base de datos de Proyectalia deberíamos introducir las siguientes instrucciones:*/
DROP DATABASE Proyectalia

/*Tenemos en la base de datos de Proyectalia una tabla llamada Empleados y queremos crear una tabla llamada
NombresEmpleados que contenga de forma única los nombres de todos los empleados:*/
SELECT DISTINCT NombreEm AS NombresEmpleados
    INTO NombreEmpleado
    FROM Empleados

/*Insertamos los nombres de los empleados (NombresEmpleados) en la tabla que tenemos para tal fin llamada
NombreEmpleado:*/
INSERT INTO NombreEmpleado (NombresEmpleados)
    SELECT DISTINCT NombreEm
    FROM Empleados

/* Para borrar la base de datos de Proyectalia deberíamos introducir las siguientes instrucciones:*/
DROP TABLE NombreEmpleado

```

```
/* Eliminar una restricción establecida sobre la tabla Proyectos para la columna CódigoEm */  
-- Creo la restricción  
ALTER TABLE Proyecto ADD  
    CONSTRAINT FK_Prueba FOREIGN KEY (CódigoEm) REFERENCES Empleados(CódigoEm)  
-- Elimino la restricción  
ALTER TABLE Proyecto  
    DROP CONSTRAINT FK_Prueba
```

```

/* -----
VARIABLES
----- */

/* Asignación con SET, para asignar un valor directamente:*/
DECLARE @Nombre1 varchar(20)
SET @Nombre1 = 'Isaac Torralba'
PRINT @Nombre1          -- Muestra el valor de la variable Nombre1 como texto.
SELECT @Nombre1         -- Muestra una tabla con el valor de la variable Nombre1.

/* Ahora con variables numéricas*/
DECLARE @a int
SET @a=7
SELECT @a

/* Asignación con SELECT, para asignar un valor consultado de la tabla socios:*/
DECLARE @Nombre2 varchar(50)
SELECT @Nombre2 = NombreEm FROM Empleados WHERE CódigoEm = 2
PRINT @Nombre2          -- Muestra el valor de la variable Nombre2 como texto.

/* Asignación con SELECT, para asignar el número de socios que disponemos a una variable:*/
DECLARE @NúmEmpleados int
SELECT @NúmEmpleados = Count (*) FROM Empleados
PRINT 'Hay ' + Cast(@NúmEmpleados AS varchar) + ' empleados.'
-- o bien
PRINT 'Hay ' + CONVERT(varchar, @NúmEmpleados) + ' Empleados.'
-- Ambas muestran el texto 'Hay 8 socios.'

/* Conversiones mediante cast*/
DECLARE @b char(1)
SELECT @b='5'
PRINT @b + ' -->' + Cast(@a AS varchar(1))
PRINT Cast(@b AS int) + @a

/* Pasar valores a una consulta o instrucción */
-- Carga la variable
DECLARE @Nombre3 varchar(20)
SET @Nombre3 = 'Isaac Torralba'
-- Realiza la consulta
SELECT *
      FROM Socios
      WHERE NombreSoc = @Nombre3

/* Utilización de variables globales*/
SELECT @@VERSION          AS 'Versión de SQL Server',
       @@SERVERNAME       AS 'Nombre del servidor',
       @@SERVICENAME      AS 'Nombre del servicio'

PRINT @@SERVERNAME

/* Conversión de un valor varchar a datetime.*/
DECLARE @fecha1 varchar(20)
SET @fecha1 = Convert (datetime, '11/12/2008', 103)          -- El 103 indica el formato en el que está escrita la
fecha (dd/mm/aa)
PRINT @fecha1          -- Muestra el valor de la variable fecha1 como datetime.

/* Convertimos ahora una fecha a varchar y la formateamos. El 3 indica el formato dd/mm/aa */
DECLARE @fecha2 datetime, @fechaFormateada varchar(20)
SET @fecha2 = GetDate ()
SET @fechaFormateada = Convert (varchar(20), @fecha2, 3)
PRINT @fechaFormateada

```

```
/* Un ejemplo utilizando CAST, para convertir un varchar a int:*/  
DECLARE @dato1 varchar(2),          @dato2 int  
SET @dato1 = '27'  
SET @dato2 = Cast(@dato1 AS int)  
SELECT @dato2 AS Valor
```

```

/* -----
TIPOS DE DATOS
----- */

/* Crear un tipo de dato llamado Tipo1 como char(25)*/
EXEC sp_addtype Tipo1, 'char(25)', 'NOT NULL'

/* Crear un tipo de dato llamado Tipo2 como int*/
EXEC sp_addtype Tipo2, 'int', 'NULL'

/* Para borrar los tipos creados usamos*/
EXEC sp_droptype 'Tipo1'
EXEC sp_droptype 'Tipo2'

/* Crearemos un tipo de dato definido por el usuario basado en el tipo de dato "money" dándole el nombre de "ejemplo1"
que no puede ser nulo. Se hará de la siguiente forma:*/
EXEC sp_addtype ejemplo1, 'money', 'NOT NULL'

/* Crearemos un tipo de datos denominado "nss" (número de la seguridad social) que se basa en el tipo de datos varchar .
El tipo de datos "nss" se utiliza en columnas que almacenan números de la seguridad social de 11 cifras (999-99-9999). La
columna no puede ser NULL. Observe que varchar (11) */
EXEC sp_addtype nss, 'varchar (11)', 'NOT NULL'

/* Crearemos un tipo de datos (basado en el tipo de datos datetime) denominado "cumpleaños" que permite valores NULL.
*/
EXEC sp_addtype cumpleaños, 'datetime', 'NULL'

/* Crearemos dos tipos de datos, "telefono" y "fax", para números de teléfono y fax. */
EXEC sp_addtype telefono, 'varchar (24)', 'NOT NULL'
EXEC sp_addtype fax, 'varchar (24)', 'NULL'

/* Crea una tabla llamada empleados2 dentro de la base de datos de Proyectalia, en la que usaremos los datos definidos
con anterioridad:*/
CREATE TABLE Empleados2 (
    NumSegSoc      nss      PRIMARY KEY,
    DNIEm2 char (9) CHECK (DNIEm2 LIKE
        '[0-9A-Z][0-9][0-9][0-9][0-9][0-9][0-9][0-9A-Z]')
        UNIQUE NOT NULL,
    Teléfono telefono,
    FAX            fax,
    FechaNac      cumpleaños,
)
GO

/* Para borrar la tabla creada así como los tipos creados usamos*/
DROP TABLE Empleados2
GO
EXEC sp_droptype 'ejemplo1'
EXEC sp_droptype 'nss'
EXEC sp_droptype 'cumpleaños'
EXEC sp_droptype 'telefono'
EXEC sp_droptype 'fax'

```

```

/* -----
FUNCIONES
----- */

-- FUNCIONES DEL SISTEMA -----

/* Muestra el día de la fecha */
PRINT Day(GetDate())

/* Muestra el día de una cadena de caracteres */
PRINT Day('20/12/2008')

/* Muestra el día de la semana de hoy */
PRINT DateName(dw,GetDate())

/* Para mostrar un carcter ASCII.*/
PRINT Ascii( 'a' )
PRINT Char( 97 )

/* Para mostrar la longitud de una cadena.*/
PRINT Len( 'Buenos días' )

/* Para reemplaza las 'o' de 'Hola' por 'e'.*/
PRINT Replace( 'Hola', 'o', 'e' )

/* Para mostrar una subcadena.*/
PRINT SubString( 'Hola', 2, 3 )

/* Para mostrar números aleatorios.*/
PRINT Rand()

/* Un ejemplo de la función IF, podría ser el que cuenta el número de socios que son 'Nombre del empleado 1' y me dice
si está o no.*/
IF (SELECT Count(*)
      FROM Empleados
      WHERE NombreEm='Nombre del empleado 1')>0
BEGIN
    PRINT 'Sí Está'
END
ELSE
BEGIN
    PRINT 'No está'
END
--O bien
IF EXISTS (SELECT Count(*)
            FROM Empleados
            WHERE NombreEm = 'Nombre del empleado 1')
BEGIN
    PRINT 'Sí Está'
END
ELSE
BEGIN
    PRINT 'No está'
END

```

```

-- FUNCIONES DEFINIDAS POR EL USUARIO -----

/* El siguiente ejemplo muestra como crear una FUNCIÓN ESCALAR. */
CREATE FUNCTION fn_MultiplicaSueldo
(
    @Id smallint,
    @Multiplicador decimal
)
RETURNS decimal (10,2)
AS
BEGIN
    DECLARE @Valor decimal,
            @Salida decimal

    SELECT @Valor = SueldoEm
           FROM Empleados
           WHERE CódigoEm = @Id

    SET @Salida = @Valor * @Multiplicador

    RETURN @Salida
END

/* Utiliza la función anteriormente creada en una sentencia. */
SELECT NombreEm, SueldoEm,
       dbo.fn_MultiplicaSueldo( CódigoEm, 10.10) AS Resultado
FROM Empleados

/* Utiliza la función anteriormente creada en una línea.*/
DECLARE @Id smallint, @Resultado decimal
SET @Id = 1
SET @Resultado = dbo.fn_MultiplicaSueldo(@Id, 10.0)
SELECT @Resultado AS Valor

/* El siguiente ejemplo muestra como crear una FUNCIÓN EN LÍNEA. */
CREATE FUNCTION fn_ProyectosEmpleado
(
    @Id smallint
)
RETURNS TABLE AS RETURN
(
    SELECT Proyecto.*
           FROM Proyecto
                INNER JOIN Empleados
                ON Empleados.CódigoEm = Proyecto.CódigoEm
           WHERE Proyecto.CódigoEm = @Id
)

/* Utiliza la función anteriormente creada en una consulta.*/
SELECT * FROM fn_ProyectosEmpleado(1)

/* Utiliza la función anteriormente creada en una consulta.*/
SELECT DNIFa, NombreEm, SueldoEm, Jefe, PresupPr, Función
FROM Familiares
    INNER JOIN Empleados
    ON Familiares.CódigoEm = Empleados.CódigoEm
    INNER JOIN fn_ProyectosEmpleado (1) AS A
    ON Empleados.CódigoEm = A.CódigoEm

```



```

/* El siguiente ejemplo muestra como crear una FUNCIÓN MULTISENTENCIA. */
CREATE FUNCTION fn_EmplDptoProy ( @Id smallint )
    RETURNS @DatosSalida TABLE (
        -- Estructura de la tabla que devuelve la función.
        CódigoEm          smallint,
        NombreEm           varchar(50),
        NombreDe           varchar(50),
        SueldoEm           smallmoney,
        Jefe               smallint,
        PresupPr           smallmoney,
        Función            varchar(50)
    )
AS
BEGIN
    -- Variables necesarias para la lógica de la función.
    DECLARE @CódigoEm smallint, @VarNombreEm varchar(50),
            @VarNombreDe varchar(50), @VarSueldoEm smallmoney,
            @VarJefe smallint, @VarPresupPr smallmoney,
            @VarFunción varchar(50)

    -- Carga los datos del empleado solicitado
    DECLARE Cdatos CURSOR FOR
        SELECT Empleados.CódigoEm, NombreEm, NombreDe, SueldoEm,
               Jefe, PresupPr, Función
        FROM Empleados
            INNER JOIN Proyecto
            ON Empleados.CódigoEm = Proyecto.CódigoEm
            INNER JOIN Departamentos
            ON Empleados.CódigoDe = Departamentos.CódigoDe
        WHERE Empleados.CódigoEm = @Id

    OPEN Cdatos

    -- Carga en Cdatos los nombres de las variables
    FETCH Cdatos
        INTO @CódigoEm , @VarNombreEm, @VarNombreDe, @VarSueldoEm,
            @VarJefe, @VarPresupPr, @VarFunción

    -- Recorremos el cursor
    WHILE (@@FETCH_STATUS = 0)
    BEGIN
        -- Cargamos en NombreDe el nombre del Departamento
        SELECT @VarNombreDe = NombreDe
        FROM Departamentos
        WHERE CódigoEm = @Id

        -- Cargamos VarPresupPr y VarFunción con el presupuesto y la función
        -- del proyecto respectivamente
        SELECT @VarPresupPr = PresupPr
        FROM Proyecto
        WHERE CódigoEm = @Id

        SELECT @VarFunción = Función
        FROM Proyecto
        WHERE CódigoEm = @Id

        -- Insertamos los datos del Empleado en la variable de salida
        INSERT INTO @DatosSalida
            (CódigoEm, NombreEm, NombreDe, SueldoEm,
             Jefe, PresupPr, Función)
        VALUES
            (@Id, @VarNombreEm, @VarNombreDe, @VarSueldoEm,
             @VarJefe, @VarPresupPr, @VarFunción)
    END

```

```
-- Vamos a la siguiente cuenta
FETCH Cdatos
      INTO    @CódigoEm , @VarNombreEm, @VarNombreDe,
              @VarSueldoEm, @VarJefe, @VarPresupPr, @VarFunción

END

-- Cierra el cursor creado
CLOSE Cdatos
DEALLOCATE Cdatos

-- Retorna los valores cargados
RETURN

END
/* Para ejecutar la función:*/
SELECT *
      FROM fn_EmplDptoProy(2)
```

```

/* -----
PROCEDIMIENTOS ALMACENADOS
----- */

/* Crea un procedimiento almacenado, denominado pa_MuestraNombre que muestra el nombre de un empleado solicitado
mediante su código sobre la tabla Empleados. */
CREATE PROCEDURE pa_MuestraNombre @Id smallint
AS
    SELECT NombreEm
    FROM Empleados
    WHERE CódigoEm = @Id
GO
--Para ejecutar este procedimiento haríamos:
EXEC pa_MuestraNombre 1
-- Borra el procedimiento almacenado
DROP PROCEDURE pa_MuestraNombre
GO

/* Muestra un procedimiento almacenado, denominado pa_AñadeEmpleado que inserta un registro en la tabla
Empleados.*/
CREATE PROCEDURE pa_AñadeEmpleado
    @DNIEmchar (9),
    @NombreEm    varchar (50),
    @DireccEm    varchar (50),
    @SueldoEm    smallmoney,
    @Jefe        smallint,
    @CódigoDe    smallint
AS
INSERT INTO Empleados (DNIEm, NombreEm, DireccEm, SueldoEm, Jefe, CódigoDe)
VALUES (@DNIEm, @NombreEm, @DireccEm, @SueldoEm, @Jefe, @CódigoDe)
GO
--Para ejecutar este procedimiento haríamos.
DECLARE @Dpto smallint
SET @Dpto = 1
EXEC pa_AñadeEmpleado '00000000A', 'Otro empleado', 'Con su dirección', 100, 1, @Dpto
-- Borra el procedimiento almacenado
DROP PROCEDURE pa_AñadeEmpleado
GO

/*El siguiente ejemplo muestra la definición de un procedimiento con parámetros de salida.*/
CREATE PROCEDURE pa_ObtenerSueldo
    @Id        smallint,
    @Sueldo smallmoney OUTPUT
AS
BEGIN
    SELECT @Sueldo = SueldoEm
    FROM Empleados
    WHERE CódigoEm = @Id
END
GO
--Y para ejecutar este procedimiento:
DECLARE @OtraVariableParaSueldo smallmoney
EXEC pa_ObtenerSueldo 1, @OtraVariableParaSueldo OUTPUT
IF @OtraVariableParaSueldo >100
    PRINT Cast(@OtraVariableParaSueldo AS varchar)
ELSE
    PRINT 'El valor es menor o igual a 100'
-- Borra el procedimiento almacenado
DROP PROCEDURE pa_ObtenerSueldo
GO

```

```
/*Muestra un procedimiento almacenado que devuelve un conjunto de resultados.*/
CREATE PROCEDURE pa_ProyectosEmpleado @Id smallint
AS
BEGIN
    SELECT Empleados.CódigoEm, DNIEm, NombreEm, PresupPr, Función
    FROM Empleados
    INNER JOIN Proyecto
    ON Empleados.CódigoEm = Proyecto.CódigoEm
    WHERE Empleados.CódigoEm = @Id
    ORDER BY PresupPr DESC
END
GO
-- La ejecución del procedimiento se realiza normalmente.
EXEC pa_ProyectosEmpleado 1
GO
-- Borra el procedimiento almacenado
DROP PROCEDURE pa_ProyectosEmpleado
GO

/* Muestra un procedimiento almacenado que devuelve un valor.*/
CREATE PROCEDURE pa_ProyectoMenorCero @Id smallint
AS
BEGIN
    IF (SELECT PresupPr
        FROM Proyecto
        WHERE CódigoPr = @Id) <= 0
        RETURN 1
    ELSE
        RETURN 0
END
GO
--Para ejecutar el procedimiento y obtener el valor devuelto.
DECLARE @ResultadoProcedimiento int
EXEC @ResultadoProcedimiento = pa_ProyectoMenorCero 1
PRINT @ResultadoProcedimiento
GO
-- Borra el procedimiento almacenado
DROP PROCEDURE pa_ProyectoMenorCero
GO
```

```
/* -----  
DISPARADORES  
----- */  
  
-- DISPARADORES DML -----  
  
/* Crea un disparador que mostrará un mensaje y se activará cuando modifique la tabla de empleados.*/  
CREATE TRIGGER TR_ModificaEmpleados  
    ON Empleados  
    FOR UPDATE  
AS  
PRINT 'Usted ha modificando la tabla empleados'  
GO  
/* Para comprobar el funcionamiento del disparador anterior podemos modificar cualquier registro de la tabla  
empleados.*/  
UPDATE empleados  
    SET CódigoDe = 1  
    WHERE CódigoDe = 1  
  
-- DISPARADORES DDL Sólo para versiones superiores a la del 2005-----  
  
/* La siguiente instrucción impide que se ejecuten sentencias DROP TABLE y ALTER TABLE en la base de datos. */  
CREATE TRIGGER tr_Seguridad  
    ON DATABASE  
    FOR DROP_TABLE, ALTER_TABLE  
AS  
BEGIN  
    RAISERROR ('No está permitido borrar ni modificar tablas!' , 16, 1)  
    ROLLBACK  
END
```

```

/* -----
FLUJOS DE CONTROL
----- */

-- EJEMPLOS IF ... ELSE -----

IF (SELECT Count (*)
    FROM Empleados) = 0
BEGIN
    Print 'No hay empleados en la tabla'
END
ELSE
BEGIN
    Print 'Ya existen empleados en la tabla'
END

-- EJEMPLOS CASE -----

-- Ejemplo 1
DECLARE @Valor int
SELECT @Valor = Count (*)
    FROM Empleados
Print CASE @Valor
    WHEN '0' THEN
        'No hay empleados en la tabla'
    WHEN '1' THEN
        'Existe un empleado en la tabla'
    ELSE
        'Existen más de un empleado en la tabla'
END

-- Ejemplo 2
DECLARE @Completo varchar(100), @Abreviado varchar(3)
SET @Abreviado = 'ITV'
SET @Completo = (CASE @Abreviado
    WHEN 'ITV' THEN 'www.ItvSoft.net'
    WHEN 'FPC' THEN 'www.ItvSoft.es'
    ELSE 'www.google.es'
END)
PRINT @Completo

-- Ejemplo 3
DECLARE @Completo varchar(100), @Abreviado varchar(3)
SET @Abreviado = 'ITV'
SET @Completo =(CASE
    WHEN @Abreviado = 'ITV' THEN (SELECT NombreEm
                                FROM Empleados
                                WHERE CódigoEm=1)
    WHEN @Abreviado = 'FPC' THEN (SELECT NombreEm
                                FROM Empleados
                                WHERE CódigoEm=2)
    ELSE 'Ninguno'
END)
PRINT @Completo

```

```
-- EJEMPLOS WHILE -----

-- Ejemplo 1
DECLARE @Contador int
SET @Contador = 0
WHILE (@Contador < 10)
    BEGIN
        SET @Contador = @Contador + 1
        PRINT 'Iteracion del bucle ' + Cast (@Contador AS varchar)
    END

-- Podemos pasar a la siguiente iteración del bucle utilizando CONTINUE.
DECLARE @Contador int
SET @Contador = 0
WHILE (@Contador < 10)
    BEGIN
        SET @Contador = @Contador + 1
        IF (@Contador % 2 = 0)
            CONTINUE
        PRINT 'Iteracion del bucle ' + Cast (@Contador AS varchar)
    END

-- El bucle se dejará de repetir con la instrucción BREAK.
DECLARE @Contador int
SET @Contador = 0
WHILE (1 = 1)
    BEGIN
        SET @Contador = @Contador + 1
        IF (@Contador % 5 = 0)
            BREAK
        PRINT 'Iteracion del bucle ' + Cast (@Contador AS varchar)
    END

-- También podemos utilizar el bucle WHILE conjuntamente con subconsultas.
DECLARE @Contador int
SET @Contador = 0
DECLARE @PresupuestoMáximo smallmoney, @PresupuestoProyecto smallmoney
SET @PresupuestoMáximo = (SELECT Max(PresupPr)
                        FROM Proyecto)
WHILE EXISTS (SELECT *
              FROM Proyecto
              WHERE PresupPr <> @PresupuestoMáximo)
    -- La subconsulta se ejecuta una vez por cada proyecto
    BEGIN
        SET @Contador = @Contador + 1
        UPDATE Proyecto
            SET PresupPr = @PresupuestoMáximo
        PRINT 'Presupuestos actualizados a: ' +
            Cast (@PresupuestoMáximo AS varchar)
    END
END
```

## 2) Introducción

### a) ¿Qué es?

Las aplicaciones en red son cada día más numerosas y versátiles. En muchos casos, el esquema básico de operación es una serie de scripts que rigen el comportamiento de una base de datos.

Debido a la diversidad de lenguajes y de bases de datos existentes, la manera de comunicar entre unos y otras sería realmente complicada a gestionar de no ser por la existencia de estándares que nos permiten el realizar las operaciones básicas de una forma universal.

Es de eso de lo que trata el Structured Query Language que no es más que un lenguaje estándar de comunicación con bases de datos. Hablamos por tanto de un lenguaje normalizado que nos permite trabajar con cualquier tipo de lenguaje (ASP o PHP) en combinación con cualquier tipo de base de datos (MS Access, SQL Server, MySQL...).

El hecho de que sea estándar no quiere decir que sea idéntico para cada base de datos. En efecto, determinadas bases de datos implementan funciones específicas que no tienen necesariamente que funcionar en otras.

Aparte de esta universalidad, el SQL posee otras dos características muy apreciadas. Por una parte, presenta una potencia y versatilidad notables que contrasta, por otra, con su accesibilidad de aprendizaje.

Por supuesto, a partir del estándar cada sistema ha desarrollado su propio SQL que puede variar de un sistema a otro, pero con cambios que no suponen ninguna complicación para alguien que conozca un SQL concreto.

### b) ¿Para qué sirve?

SQL es un lenguaje de programación que permite crear para después interrogar a bases de datos relacionales.

Este lenguaje permite crear tablas y relaciones para después realizar complejos sistemas de búsqueda.

Es por ello que SQL es el lenguaje más utilizado por los principales sistemas de gestión de datos lo cual es una prueba de su valía.

SQL se utiliza para crear objetos QueryDef, como el argumento de origen del método OpenRecordSet y como la propiedad RecordSource del control de datos. También se puede utilizar con el método Execute para crear y manipular directamente las bases de datos Jet y crear consultas SQL de paso a través para manipular bases de datos remotas cliente - servidor.

### c) Subtipos del lenguaje: DDL, DML, DCL.

El DDL (Data Description Language), lenguaje de definición de datos, incluye órdenes para definir, modificar o borrar las tablas en las que se almacenan los datos y de las relaciones entre estas. (Es el que más varía de un sistema a otro).



Un lenguaje de definición de datos (Data Definition Language, DDL por sus siglas en inglés) es un lenguaje proporcionado por el [sistema de gestión de base de datos](#) que permite a los usuarios de la misma llevar a cabo las tareas de definición de las estructuras que almacenarán los datos así como de los procedimientos o funciones que permitan consultarlos.

El lenguaje de programación SQL, el más difundido entre los gestores de bases de datos, admite las siguientes sentencias de definición: CREATE, DROP y ALTER, cada una de las cuales se puede aplicar a las tablas, vistas, procedimientos almacenados y disparadores de la [base de datos](#).

Otras que se incluyen dentro del DDL, pero que su existencia depende de la implementación del estándar SQL que lleve a cabo el [gestor de base de datos](#) son GRANT y REVOKE, los cuales sirven para otorgar permisos o quitarlos, ya sea a usuarios específicos o a un rol creado dentro de la [base de datos](#).

Lenguaje de Definición de Datos (DDL): permite establecer y/o modificar el esquema relacional, es decir, añadir, borrar o actualizar atributos, tablas, índices, etc.

El **DML** (Data Manipulation Language), lenguaje de manipulación de datos, nos permite recuperar los datos almacenados en la base de datos y también incluye órdenes para permitir al usuario actualizar la base de datos añadiendo nuevos datos, suprimiendo datos antiguos o modificando datos previamente almacenados.

El lenguaje de manipulación de datos más popular hoy día es [SQL](#), usado para recuperar y manipular datos en una [base de datos relacional](#). Otros ejemplos de DML son los usados por [bases de datos IMS/DL1](#), [CODASYL](#) u otras.

Lenguaje de Manipulación de Datos (DML): permite manipular los datos del esquema relacional, es decir, consultar, actualizar, o borrar información.

El **DCL** (Data Control Language), lenguaje de control de datos, contiene elementos útiles para trabajar en un entorno multiusuario, en el que es importante la protección de los datos, la seguridad de las tablas y el establecimiento de restricciones en el acceso, así como elementos para coordinar cómo se compartirán los datos por parte de usuarios concurrentes, asegurando que no interfieren unos con otros.

Algunos ejemplos de comandos incluidos en el DCL son los siguientes:

- GRANT: Permite dar permisos a uno o varios usuarios o roles para realizar tareas determinadas.
- REVOKE: Permite eliminar permisos que previamente se han concedido con GRANT.

Las tareas sobre las que se pueden conceder o denegar permisos son las siguientes:

- CONNECT
- SELECT
- INSERT
- UPDATE
- DELETE
- USAGE

#### d) Componentes

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

#### i) Comandos

Existen dos tipos de comandos SQL:

- DDL que permiten crear y definir nuevas bases de datos, campos e índices.
- DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

##### Comandos DDL

Comando	Descripción
CREATE	Utilizado para <b>crear</b> nuevas tablas, campos e índices
DROP	Empleado para <b>eliminar</b> tablas e índices
ALTER	Utilizado para <b>modificar</b> las tablas agregando campos o cambiando la definición de los campos.

##### Comandos DML

Comando	Descripción
SELECT	Utilizado para <b>consultar</b> registros de la base de datos que satisfagan un criterio determinado
INSERT	Utilizado para <b>cargar</b> lotes de datos en la base de datos en una única operación.
UPDATE	Utilizado para <b>modificar</b> los valores de los campos y registros especificados
DELETE	Utilizado para <b>eliminar</b> registros de una tabla de una base de datos

## ii) Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Cláusula	Descripción
FROM	Utilizada para <b>especificar la tabla</b> de la cual se van a seleccionar los registros
WHERE	Utilizada para <b>especificar las condiciones</b> que deben reunir los registros que se van a seleccionar
GROUP BY	Utilizada para <b>separar los registros</b> seleccionados en grupos específicos
HAVING	Utilizada para expresar la <b>condición</b> que debe satisfacer cada grupo
ORDER BY	Utilizada para <b>ordenar</b> los registros seleccionados de acuerdo con un orden específico

## iii) Operadores lógicos

Operador	Uso
AND	Es el "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Es el "o" lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

## iv) Operadores de comparación

Operador	Uso
<	Menor que.
>	Mayor que.
<>	Distinto de.
<=	Menor o igual que.
>=	Mayor o igual que.
=	Igual que.
BETWEEN	Utilizado para especificar un intervalo de valores.
LIKE	Utilizado en la comparación de un modelo.
In	Utilizado para especificar registros de una base de datos.

## v) Funciones de agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

Función	Descripción
AVG	Utilizada para calcular el <b>promedio</b> de los valores de un campo determinado.
COUNT	Utilizada para devolver el <b>número de registros</b> de la selección.
SUM	Utilizada para devolver la <b>suma</b> de todos los valores de un campo determinado.
MAX	Utilizada para devolver el <b>valor más alto</b> de un campo especificado.
MIN	Utilizada para devolver el <b>valor más bajo</b> de un campo especificado.

## vi) Orden de ejecución de los comandos

Dada una sentencia SQL de selección que incluye todas las posibles cláusulas, el orden de ejecución de las mismas es el siguiente:

1. Cláusula FROM
2. Cláusula WHERE
3. Cláusula GROUP BY
4. Cláusula HAVING
5. Cláusula SELECT
6. Cláusula ORDER BY

### 3) Tipos de datos de SQL Server 2005 (se debe actualizar)

#### a) Tipos de datos provisto pr SQL Server

SQL Server brinda una serie de tipos de datos para almacenar la información, la correcta selección del tipo de dato es simplemente una cuestión de determinar que valores desea almacenar, como por ejemplo carácter, enteros, binario, fechas, etc.

La siguiente es una tabla que describe los tipos de datos provistos por SQL Server:

Cat.	Descripción	Tipo de Dato	Descripción
<b>Bi na ri o</b>	Almacenan cadenas de bits. La data consiste de números hexadecimales. Por ejemplo el decimal 245 es F5 en hexadecimal.	<b>binary</b>	La data debe tener una longitud fija (hasta 8 KB).
		<b>varbinary</b>	Los datos pueden variar en el número de dígitos hexadecimales (hasta 8 KB).
		<b>image</b>	La data puede tener una longitud variable y exceder los 8Kb. (Hasta 2GB)
<b>C ar ác te r</b>	Consisten de una combinación de letras, símbolos y números. Por ejemplo las combinaciones "John928" y "(O*(&(B99nhjkJ".	<b>char</b>	Los datos deben tener una longitud fija (Hasta 8 KB).
		<b>varchar</b>	La data puede variar en el número de caracteres (Hasta 8 KB.)
		<b>text</b>	Los datos pueden ser caracteres ASCII que excedan los 8 KB. (Campos memo Hasta 2GB)
<b>F ec ha y H or a</b>	Consisten en combinaciones válidas de estos datos. No puede separar en tipos distintos el almacenamiento de sólo fechas o sólo horas.	<b>Datetime</b>	Fechas en el rango 01/01/1753 hasta el 31/12/9999 (Se requieren 8 bytes por valor).
		<b>smalldatetime</b>	Fechas en el rango 01/01/1900 hasta 06/06/2079 (Se requieren 4 bytes por valor).
<b>D ec im al</b>	Consisten en información que almacena información significativa después del punto decimal.	<b>decimal</b>	Los datos pueden tener hasta 38 dígitos, todos los cuales podrían estar a la derecha del punto decimal. Este tipo de dato guarda un valor exacto del número y no una aproximación.
		<b>numeric</b>	Para SQL Server, el tipo de dato numérico es equivalente al tipo de datos decimal.
<b>Pu nt o Fl ot an te</b>	Números aproximados (Punto flotante).	<b>float</b>	Datos en el rango de $1.79E + 308$ hasta $1.79E + 308$ .
		<b>real</b>	Datos en el rango de $3.40E + 38$ hasta $3.40E + 38$ .
<b>En te ro s</b>	Consiste en información numérica positiva o negativa como por ejemplo -5, 0 y 25.	<b>bigint</b>	Datos en el rango de $2^{63}$ ( $-9223372036854775808$ ) hasta $2^{63-1}$ ( $9223372036854775807$ ). Se requieren de 8 bytes para almacenar estos valores.
		<b>int</b>	Datos en el rango de $-2,147,483,648$ hasta $2,147,483,647$ . Se requiere de 4 bytes para almacenar estos valores.
		<b>smallint</b>	Datos en el rango de $-32,768$ hasta $32,767$ . Se requieren 2 bytes por cada valor de este tipo.
		<b>tinyint</b>	Datos entre 0 y 255, se requiere de 1 byte.

Cat.	Descripción	Tipo de Dato	Descripción
<b>Monetario</b>	Cantidades monetarias positivas o negativas.	<b>money</b>	Datos monetarios entre - 922,337,203,685,477.5808 y +922,337,203,685,477.5807 (Se requieren 8 bytes por valor).
		<b>smallmoney</b>	Datos monetarios entre - 214,748.3648 y 214,748.3647 (Se requieren de 4 bytes por valor).
<b>Especial</b>	Consisten en información que no recae en ninguna de las categorías anteriormente mencionadas.	<b>bit</b>	Datos que consisten de 1 o 0. Emplear este tipo de dato para representar TRUE o FALSE ó YES o NO.
		<b>cursor</b>	Este tipo de dato es empleado por variables o procedimientos almacenados que emplean parámetros OUTPUT referenciados a un cursor.
		<b>timestamp</b>	Este tipo de dato es empleado para indicar la actividad que ocurre sobre una fila. La secuencia de este número se incrementa en formato binario.
		<b>uniqueidentifier</b>	Consiste en un número hexadecimal que especifica un globally unique identifier (GUID), es útil cuando se desea asegurar la unicidad de una fila entre muchas otras.
		<b>SQL_variant</b>	Almacena varios tipos de datos, a excepción de text, ntext, timestamp, image y sql_variant.
		<b>table</b>	Almacena un resultado de una consulta para su posterior procesamiento. Se puede emplear para definir variables locales de tipo table o para retornar los valores devueltos por una función del usuario.
<b>Unicode</b>	Al emplear este tipo de datos se puede almacenar sobre una columna valores que incluyan este conjunto de caracteres. Hay que recordar que los datos Unicode emplean dos bytes por cada carácter a representar.	<b>nchar</b>	Datos con longitud fija, hasta 4000 caracteres Unicode.
		<b>nvarchar</b>	Datos que pueden variar, hasta 4000 caracteres Unicode.
		<b>ntext</b>	Datos que exceden los 4000 caracteres Unicode.

## b) Tipos de datos definidos por el usuario

Los tipos de datos definidos por el usuario están basados en tipos predefinidos por el sistema en SQL Server 2005.

Tanto los tipos de datos del sistema como los definidos por el usuario son usados para asegurar la integridad de datos.

Los tipos de datos definidos por el usuario pueden ser usados en varias tablas que deban guardar el mismo tipo de dato en una columna y cuando se necesita asegurar que estas columnas tengan exactamente el mismo tipo de dato, longitud y capacidad de aceptar nulos.

Por ejemplo, un tipo de datos definido por el usuario llamado CódigoPostal podría ser creado en base al tipo char.

Cuando se crea un tipo de dato definido por el usuario, se deben proveer los siguientes parámetros:

- Nombre
- Tipo de datos del sistema sobre el que se basa el nuevo tipo de dato
- Anulabilidad (si el tipo de dato permite valores nulos).

Cuando la anulabilidad no es explícitamente definida, se toma por defecto la configuración de nulos ANSI para la base de datos o conexión.

Si un tipo de datos definido por el usuario es creado en la base de datos MODEL el tipo de datos estará disponible para todas las nuevas bases de datos que se creen. Si el tipo de datos es creado en una base de datos definida por el usuario el tipo de datos sólo estará disponible para dicha base de datos.

Se pueden crear tipos de datos definidos por el usuario utilizando el procedimiento almacenado **sp\_addtype** o utilizando el Microsoft SQL Server Management Studio.

Con el siguiente ejemplo crearemos un tipo de dato definido por el usuario basado en el tipo de dato "money" dándole el nombre de "ejemplo1" que no puede ser nulo. Se hará de la siguiente forma:

```
EXEC sp_addtype ejemplo1, money, 'NOT NULL'
```

En el ejemplo siguiente se creará un tipo de datos denominado "nss" (número de la seguridad social) que se basa en el tipo de datos varchar proporcionado por SQL Server. El tipo de datos "nss" se utiliza en columnas que almacenan números de la seguridad social de 11 cifras (999-99-9999). La columna no puede ser NULL. Observe que varchar (11) está entre comillas simples porque contiene signos de puntuación (paréntesis).

```
USE master
GO
EXEC sp_addtype nss, 'varchar (11)', 'NOT NULL'
GO
```

En este ejemplo se creará un tipo de datos (basado en el tipo de datos datetime) denominado "cumpleaños" que permite valores NULL.

```
USE master
GO
EXEC sp_addtype cumpleaños, datetime, 'NULL'
GO
```

En este ejemplo se crearán dos tipos de datos, "telefono" y "fax", para números de teléfono y fax.

```
USE master;
GO
EXEC sp_addtype telefono, 'varchar (24)', 'NOT NULL'
GO
EXEC sp_addtype fax, 'varchar (24)', 'NULL'
GO
```

La utilización de los datos definidos por el usuario se podrán ver mediante el siguiente ejemplo en el que se crea una tabla llamada empleados2 dentro de la base de datos de Proyectalia, en la que usaremos los datos definidos con anterioridad:

```
CREATE TABLE Empleados2 (
    NumSegSoc    nss                IDENTITY (1, 1)
                PRIMARY KEY,
    DNIEm2       char (9)          CHECK (DNIEm2 LIKE
                '[0-9A-Z][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9A-Z]')
                UNIQUE             NOT NULL,
    Teléfono     telefono,
    FAX           fax,
    FechaNac     cumpleaños,
)
GO
```

Para borrar la tabla creada así como los tipos creados usaremos:

```
DROP TABLE Empleados2
GO
EXEC sp_droptype 'ejemplo1'
EXEC sp_droptype 'nss'
EXEC sp_droptype 'cumpleaños'
EXEC sp_droptype 'telefono'
EXEC sp_droptype 'fax'
```

En el ejemplo siguiente se crea un tipo de alias basado en el tipo de datos varchar suministrado por el sistema.

```
CREATE TYPE SSN
FROM          varchar(11)          NOT          NULL          ;
```

#### 4) Constantes: cómo se expresan las cadenas de caracteres, los enteros, las fechas, las horas,...

Una constante es un símbolo que representa el valor específico de un dato. El formato de las constantes depende del tipo de datos del valor que representa. Las constantes se llaman también literales.

El formato de las constantes depende del tipo de datos del valor que representan.

##### Constantes de cadena de caracteres

Las constantes de cadena de caracteres van entre comillas simples e incluyen caracteres alfanuméricos (a-z, A-Z y 0-9) y caracteres especiales, como el signo de exclamación (!), la arroba (@) y el signo de número (#). A las constantes de cadena de caracteres se les asigna la intercalación predeterminada de la base de datos actual, a menos que se utilice la cláusula *COLLATE* para especificar una intercalación. Las cadenas de caracteres escritas por los usuarios se evalúan a través de la página de códigos del equipo y, si es necesario, se traducen a la página de códigos predeterminada de la base de datos.

Si una cadena de caracteres entre comillas simples contiene una comilla, represente la comilla interna con dos comillas simples. Esto no es necesario en las cadenas incluidas entre comillas dobles.

Éstos son algunos ejemplos de cadenas de caracteres:

```
'55555555E' 'José Leño Denso' 'Dirección_5' 'O'
'Tu id es: %d debería estar entre %d y %d.' '' (Cadena vacía)
```

Éste es un ejemplo de cadena de caracteres Unicode:

```
N'Camión'
```

##### Constantes binarias

Las constantes binarias tienen el prefijo 0x y son cadenas de números hexadecimales. No se incluyen entre comillas.

Éstos son algunos ejemplos de cadenas binarias:

```
0xAE 0x12Ef0x69048 0x (número vacío 0)
```

##### Constantes bit

Las constantes de tipo bit se representan con los números 0 ó 1, y no se incluyen entre comillas. Si se utiliza un número mayor que uno, se convierte en uno.

Ejemplos:

```
Declarando un atributo BIT puedes poner que una luz este encendida 1 o apagada 0 o el
sexo de un empleado 1 para varones y 0 para mujeres.
```



**Constantes datetime**

Las constantes de tipo datetime se representan mediante valores de fecha en formatos específicos de cadenas de caracteres, incluidos entre comillas simples. Para obtener más información acerca de los formatos de las constantes datetime, vea Utilizar datos de fecha y hora.

Éstos son algunos ejemplos de constantes datetime:

'Abril 15, 2008'      '15 Abril, 2008'      '980415'      '04/15/2008'

Ejemplos de constantes de hora:

'14:30:24'      '04:24 PM'

**Constantes integer**

Las constantes de tipo integer se representan mediante una cadena de números sin comillas y sin separadores decimales. Las constantes integer deben ser números enteros y no pueden contener decimales.

Éstos son algunos ejemplos de constantes integer:

1894      2

**Constantes decimal**

Las constantes de tipo decimal se representan mediante una cadena de números sin comillas y con separador decimal.

Éstos son algunos ejemplos de constantes decimal:

184.124      2.0

**Constantes float y real**

Las constantes de tipo float y real se representan en notación científica.

Éstos son algunos ejemplos de valores de tipo float o real:

11.5E5      0.5E-2

**Constantes money**

Las constantes de tipo money se representan como una cadena de números con un separador decimal y un símbolo de moneda opcionales como prefijo. Las constantes money no van entrecomilladas.

SQL Server 2005 no impone ninguna clase de reglas de agrupamiento, como la inserción de una coma (,) cada tres dígitos, en las cadenas que representan dinero.

Éstos son algunos ejemplos de constantes money:

€12      €54223.14

**Constantes uniqueidentifier**

Las constantes de tipo uniqueidentifier son una cadena que representa un GUID. Se pueden especificar en formato de cadena de caracteres o binario.

Estos dos ejemplos especifican el mismo GUID:

'6F9619FF-8B86-D011-B42D-00C04FC964FF'  
0xff19966f868b11d0b42d00c04fc964ff

**Características comunes:****Especificar números negativos y positivos**

Para indicar si un número es positivo o negativo, aplique el operador unario + o - a una constante numérica. Esto crea una expresión numérica que representa el valor numérico con signo. Las constantes numéricas son positivas si no se aplica el operador unario + o -.

Expresiones integer con signo:

+145345234	-2147483648
------------	-------------

Expresiones decimal con signo:

+145345234.2234	-2147483648.10
-----------------	----------------

Expresiones float con signo:

+123E-3	-12E5
---------	-------

Expresiones money con signo:

-€45.56	+€423456.99
---------	-------------

**Utilizar constantes en Transact-SQL**

Las constantes se pueden usar de muchas formas en Transact-SQL. A continuación se muestran algunos ejemplos:

- Como un valor constante de una expresión aritmética:

```
SELECT SueldoEm + €1.10
FROM Empleados
```

- Como el valor con el que se compara una columna en una cláusula WHERE:

```
SELECT *
FROM Empleados
WHERE NombreEm = 'Isaac Torralba'
```

- Como el valor que se va a colocar en una variable:

```
SET @IVA = 16.00
```

- Como el valor que debe colocarse en una columna de la fila actual. Esto se especifica con la cláusula SET de la instrucción UPDATE o la cláusula VALUES de una instrucción INSERT:

```
UPDATE Empleados
SET SueldoEm = €99.99
WHERE CódigoDe = 1
```

- Con la cláusula VALUES de una instrucción INSERT:

```
INSERT INTO Familiares (DNIFa, CódigoEm)
VALUES ('20050000B', 2)
```

- Como la cadena de caracteres que especifica el texto del mensaje emitido por una instrucción PRINT o RAISERROR:

```
PRINT 'Esto es un mensaje.'
```

- Como el valor que se va a probar en una instrucción condicional, como, por ejemplo, una instrucción IF o funciones CASE:

```
IF (@VentasTotales > €100000.00) EXECUTE PR_Almacén1
```

## 5) Creación, modificación y eliminación de objetos (BD, Tablas, Vistas, Índices,...)

### a) CREATE

SQL ServerTransact-SQL contiene las instrucciones CREATE que se utilizan para definir nuevas entidades. Por ejemplo, se utiliza CREATE TABLE para agregar una nueva tabla a una base de datos.

#### i) Base de datos

Crea una nueva base de datos y los archivos que se utilizan para almacenar la base de datos o adjunta una base de datos desde los archivos de una base de datos creada anteriormente.

En SQL Server una BD se crea de la siguiente forma:

```
CREATE DATABASE <nombre_BD>,
```

Pidiéndonos posteriormente información sobre su tamaño, ubicación de ficheros,..., por ejemplo:

```
/*=====
PROYECTALIA
=====
Creación de la Base de Datos
===== */

-- Pone en uso la base de datos Master
USE Master
GO

-- Comprobación, si existe la borra.
IF EXISTS (SELECT * FROM master..sysdatabases
           WHERE name = N'Proyectalia')
BEGIN
    -- Borra la Base de datos y la crea la base de datos.
    DROP DATABASE Proyectalia
    CREATE DATABASE Proyectalia
END
ELSE
BEGIN
    -- Crea la base de datos
    CREATE DATABASE Proyectalia
END
GO

-- Pone en uso la base de datos
USE Proyectalia
GO
```

## ii) Tablas:

Crea una nueva Tabla. Una de las principales sentencias de definición de datos es la de creación de una tabla. Su sintaxis es la siguiente:

```
CREATE TABLE tabla (atributo tipo)
```

La ejecución de esta sentencia, genera como resultado una nueva tabla, en la base de datos en la que estemos conectados.

- El nombre de la tabla debe ir después de la palabra TABLE.
- El nombre de los atributos deberá ir entre paréntesis, especificando el tipo. Si existe más de un atributo, se deberán separar por comas.

Por ejemplo vamos a crear la tabla de empleados dentro de la base de datos de Proyectalia

```
CREATE TABLE Empleados (
    CódigoEm smallint IDENTITY (1, 1)
    PRIMARY KEY,
    DNIEm char (9) CHECK (DNIEm LIKE
        '[0-9A-Z][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9A-Z]')
    UNIQUE NOT NULL,
    NombreEm varchar (50) NOT NULL,
    DireccEm varchar (50) NOT NULL,
    SueldoEm smallmoney NOT NULL,
    Jefe smallint NULL,
    CódigoDe smallint DEFAULT 1
    -- Necesita modificaciones (al final).
    -- Clave ajena (Jefe, CódigoDe)
)
GO
```

### (1)Tipos de datos de las columnas

Los tipos de datos pueden ser clasificados atendiendo al contenido, de la siguiente forma:

#### Datos numéricos exactos

bigint	numeric	bit	smallint	decimal
smallmoney	int	tinyint	money	

#### Datos numéricos aproximados

Float	real
-------	------

#### Datos de fecha y hora

Date	datetimeoffset	datetime2	smalldatetime	datetime
Time				

#### Datos de cadenas de caracteres

char	varchar	text
------	---------	------

#### Datos de cadenas de caracteres unicode

nchar	nvarchar	ntext
-------	----------	-------

#### Datos de cadenas binarias

binary	varbinary	image
--------	-----------	-------

**Otros tipos de datos**

cursor                      timestamp                      hierarchyid                      uniqueidentifier                      sql\_variant  
xml                      table

Los tipos de datos son los explicados con anterioridad, pero no deberíamos dejar de ver las funciones que están asociadas con la manipulación de Fechas

Función	Determinismo
DateAdd	Devuelve un valor datetime nuevo que se basa en la suma de un intervalo a la fecha especificada.
DateDiff	Devuelve el número de límites de fecha y hora que hay entre dos fechas especificadas.
DateTime	Devuelve una cadena de caracteres que representa la parte de la fecha especificada de la fecha especificada.
DatePart	Devuelve un entero que representa la parte de la fecha especificada de la fecha indicada.
Day	Devuelve un entero que representa la parte del día de la fecha especificada.
GetDate	Devuelve la fecha y hora actuales del sistema en el formato interno estándar de Microsoft® SQL Server™ para los valores datetime.
GetUTVDate	Devuelve el valor de datetime que representa la hora UTC actual (Universal Coordinated Time u hora del meridiano de Greenwich). La hora UTC actual se deriva de la hora local actual y la configuración de zona horaria del sistema operativo del equipo en el que se ejecuta SQL Server.
Month	Devuelve un entero que representa el mes de una fecha especificada.
Year	Devuelve un entero que representa la parte de año de la fecha especificada.

**(2) Restricciones****(a) PRIMARY KEY**

Genera un índice primario el campo o los campos especificados.

Este ejemplo muestra una columna llamada CódigoPr de la tabla Proyecto, indicando que el Código del Proyecto identificará de manera única a cada elemento de la tabla Proyectos.

CódigoPr int PRIMARY KEY

Observaciones:

- Una tabla puede contener una sola restricción PRIMARY KEY.
- El índice generado por una restricción PRIMARY KEY no puede hacer que el número de índices de la tabla exceda de 249 índices no agrupados y 1 índice agrupado.
- Si no se especifica CLUSTERED o NONCLUSTERED para una restricción PRIMARY KEY, se utiliza CLUSTERED si no hay índices agrupados especificados para las restricciones UNIQUE.

- Todas las columnas definidas en una restricción PRIMARY KEY deben establecerse como NOT NULL. Si no se especifica la posibilidad de aceptar NULL, todas las columnas que participan en una restricción PRIMARY KEY tienen su posibilidad de aceptar NULL establecida en NOT NULL.

## (b) UNIQUE

Las restricciones UNIQUE se utilizan para definir un índice único en las columnas de claves no principales. Una columna de restricción PRIMARY KEY incluye automáticamente una restricción de unicidad; sin embargo, una restricción UNIQUE puede aceptar valores NULL.

Un índice único es un índice que no permite valores duplicados, es decir que si una columna tiene definida una restricción de UNIQUE no podrán haber dos filas con el mismo valor en esa columna. Se suele emplear para que el sistema compruebe el mismo que no se añaden valores que ya existen, por ejemplo si en una tabla de clientes queremos asegurarnos que dos clientes no puedan tener el mismo DNI y la tabla tiene como clave principal un código de cliente, definiremos la columna DNI con la restricción de UNIQUE.

Este ejemplo muestra una columna llamada DNIEm de la tabla Empleados. Exige la restricción de que el DNI de la tabla empleados sea único.

```
DNIEm char (9) UNIQUE NOT NULL
```

El ejemplo siguiente muestra una restricción UNIQUE creada en las columnas CódigoEm y DireccEm de la tabla Empleados, donde CódigoEm es actualmente la restricción PRIMARY KEY; no debe haber dos Empleados iguales en la misma Dirección.

```
CONSTRAINT U_CodDir UNIQUE (CódigoEm y DireccEm)
```

Observaciones:

- Cada restricción UNIQUE genera un índice. El número de restricciones UNIQUE no puede hacer que el número de índices de la tabla exceda de 249 índices no agrupados y 1 índice agrupado.

### (c) NOT NULL

La cláusula NOT NULL indica que la columna no podrá contener un valor nulo, es decir que se deberá rellenar obligatoriamente y con un valor válido.

El ejemplo siguiente muestra una restricción NOT NULL creada sobre la columna SueldoEm y una restricción NULL (por defecto, con lo cual no sería necesario expresarla) sobre la columna Jefe de la tabla Empleados, donde SueldoEm no puede contener valores nulos y Jefe sí podría tener valores nulos.

```
SueldoEm smallmoney NOT NULL,  
Jefe smallint NULL
```

### (d) FOREIGN KEY

Es una restricción que proporciona integridad referencial para los datos de la columna o columnas. Las restricciones FOREIGN KEY requieren que cada valor de la columna exista en la columna de referencia correspondiente de la tabla a la que se hace referencia. Las restricciones FOREIGN KEY pueden hacer referencia sólo a columnas que sean restricciones PRIMARY KEY o UNIQUE en la tabla de referencia o a columnas a las que se haga referencia en UNIQUE INDEX en la tabla de referencia.

El formato para introducir una restricción de tipo FOREIGN KEY es el siguiente:

```
[CONSTRAINT nombre] FOREIGN KEY [id] (nombre_índice, ...)  
REFERENCES nombre_de_tabla (nombre_índice, ...)  
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]  
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

Las restricciones FOREIGN KEY se pueden:

- Crear cuando se crea la tabla, durante el proceso de definición de la misma.
- Agregar a una tabla ya existente, siempre que la restricción FOREIGN KEY esté vinculada a una restricción PRIMARY KEY o UNIQUE de otra o de la misma tabla. Una tabla puede contener varias restricciones FOREIGN KEY.
- Modificar o eliminar si ya existen restricciones FOREIGN KEY. Por ejemplo, es posible que desee que la restricción FOREIGN KEY de la tabla haga referencia a otras columnas. No se puede cambiar la longitud de una columna definida con una restricción FOREIGN KEY.



El ejemplo siguiente muestra una restricción FOREIGN KEY creada sobre la columna CódigoEm de la tabla Familiares que no nos permite introducir un Código de empleado si previamente el empleado no existe.

*Dentro de la instrucción CREATE TABLE*  
CódigoEm smallint REFERENCES Empleados (CódigoEm)

*Modificando la estructura de la tabla:*

```
ALTER TABLE Empleados ADD  
    CONSTRAINT FK_CódigoEm FOREIGN KEY (CódigoEm)  
    REFERENCES Empleados (CódigoEm)
```

El ejemplo siguiente muestra una restricción FOREIGN KEY creada sobre la columna IdPelícula de la tabla Ejemplares que nos permite modificar y eliminar Código en cascada.

```
ALTER TABLE Ejemplares ADD  
    CONSTRAINT FK_Ejemplares FOREIGN KEY (IdPelícula)  
    REFERENCES Películas (IdPelícula)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE
```

Observaciones:

- Cuando en la columna de una restricción FOREIGN KEY se introduce un valor distinto de NULL, el valor debe existir en la columna a la que se hace referencia; de lo contrario, se devuelve un mensaje de error de infracción de clave externa.
- Las restricciones FOREIGN KEY se aplican a la columna anterior a menos que se especifiquen columnas de origen.
- Las restricciones FOREIGN KEY sólo pueden hacer referencia a las tablas de la misma base de datos en el mismo servidor. La integridad referencial de las bases de datos cruzadas debe implementarse a través de desencadenadores.
- Las restricciones FOREIGN KEY pueden hacer referencia a otras columnas de la misma tabla (una autoreferencia).
- La cláusula REFERENCES de una restricción FOREIGN KEY en el nivel de columna puede enumerar sólo una columna de referencia, que debe tener el mismo tipo de datos que la columna en la que se define la restricción.
- La cláusula REFERENCES de una restricción FOREIGN KEY en el nivel de tabla debe tener el mismo número de columnas de referencia que el número de columnas de la lista de columnas de restricción. El tipo de datos de cada columna de referencia debe ser también el mismo que la columna correspondiente de la lista de columnas.



- Es posible que no se pueda especificar **CASCADE** si una columna del tipo **timestamp** forma parte de la clave externa o de la clave a la que se hace referencia.
- Se puede combinar **CASCADE** y **NO ACTION** en tablas que tengan relaciones referenciales entre sí. Si **SQL Server** encuentra **NO ACTION**, termina y deshace las acciones **CASCADE** relacionadas. Cuando una instrucción **DELETE** hace que se combinen acciones **CASCADE** y **NO ACTION**, todas las acciones **CASCADE** se aplican antes de que **SQL Server** compruebe si hay cláusulas **NO ACTION**.
- Una tabla puede contener un máximo de 253 restricciones **FOREIGN KEY**.
- Las restricciones **FOREIGN KEY** no se exigen en tablas temporales.
- Una tabla puede hacer referencia a un máximo de 253 tablas distintas en sus restricciones **FOREIGN KEY**.
- Las restricciones **FOREIGN KEY** sólo pueden hacer referencia a las columnas de las restricciones **PRIMARY KEY** o **UNIQUE** de la tabla de referencia o a las columnas en **UNIQUE INDEX** de la tabla de referencia.

#### (i) ON DELETE

Especifica qué acción tiene lugar en una fila de la tabla creada, si esa fila tiene una relación referencial y la fila a la que hace referencia se elimina en la tabla primaria. El valor predeterminado es **NO ACTION**.

Si se especifica **CASCADE** y se elimina una fila de la tabla primaria, también se elimina la fila de la tabla desde donde se hace referencia. Si se especifica **NO ACTION**, **SQL Server** genera un error y se deshace la acción de eliminación de la fila en la tabla primaria.

Por ejemplo, en la base de datos **Proyectalia**, la tabla **Familiares** tiene una relación de integridad referencial con la tabla **Empleados**. La clave externa **CódigoEm** de la tabla **Familiares** hace referencia a la clave principal **CódigoEm** de **Empleados**.

```
CódigoEm smallint REFERENCES Empleados (CódigoEm)  
ON DELETE CASCADE
```

Si se ejecuta una instrucción **DELETE** en una fila de la tabla **Empleados** y se especifica la acción **ON DELETE CASCADE** para el **CódigoEm** de los **Familiares**, **SQL Server** comprueba las filas dependientes de la tabla **Familiares**. Si las hay, las filas dependientes de la tabla **Familiares** se eliminan, así como la fila a la que se hace referencia en la tabla **Empleados**.

Por otra parte, si se especifica **NO ACTION**, SQL Server genera un error y deshace la acción de eliminación de la fila Empleados si al menos hay una fila en la tabla Familiares que haga referencia a la fila Empleados.

El valor predeterminado es **NO ACTION**.

- **NO ACTION**: El Motor de base de datos genera un error y se revierte la acción de eliminación de la fila de la tabla primaria.
- **CASCADE**: Se eliminan las filas correspondientes de la tabla a la que se hace referencia si la fila se elimina de la tabla primaria.
- **SET NULL**: Todos los valores que forman la clave externa se establecen en **NULL** si se elimina la fila correspondiente de la tabla primaria. Para ejecutar esta restricción, las columnas de clave externa deben admitir valores **NULL**.
- **SET DEFAULT**: Todos los valores que forman la clave externa se establecen en los valores predeterminados si se elimina la fila correspondiente de la tabla primaria. Para ejecutar esta restricción, las columnas de clave externa deben tener valores predeterminados. Si la columna admite valores **NULL** y no hay ningún valor predeterminado establecido de forma explícita, **NULL** se convierte en el valor predeterminado implícito de la columna.

## (ii) **ON UPDATE**

Especifica qué acción tiene lugar en una fila de la tabla creada, si esa fila tiene una relación referencial y la fila a la que hace referencia se actualiza en la tabla primaria. El valor predeterminado es **NO ACTION**.

Si se especifica **CASCADE**, la fila se actualiza en la tabla de referencia si esa fila se actualiza en la tabla primaria. Si se especifica **NO ACTION**, SQL Server genera un error y se deshace la acción de actualización en la fila de la tabla primaria.

Por ejemplo, en la base de datos Proyectalia, la tabla Familiares tiene una relación referencial con la tabla Empleados: la clave externa CódigoEm de Familiares hace referencia a la clave principal CódigoEm de Empleados.

```
CódigoEm smallint REFERENCES Empleados (CódigoEm)
ON UPDATE CASCADE
```

Si se ejecuta una instrucción UPDATE en una fila de la tabla Empleados y se especifica la acción ON UPDATE CASCADE para CódigoEm de los Familiares, SQL Server comprueba las filas dependientes de la tabla Familiares. Si las hay, las filas dependientes de la tabla Familiares se actualizan, así como la fila a la que se hace referencia en la tabla Empleados.

Por el contrario, si se especifica NO ACTION, SQL Server emite un error y deshace la acción de actualización de la fila Empleados si al menos una fila de la tabla Familiares hace referencia a ella.

El valor predeterminado es NO ACTION.

- NO ACTION: El Motor de base de datos genera un error y se revierte la acción de actualización de la fila de la tabla primaria.
- CASCADE: Se actualizan las filas correspondientes de la tabla a la que se hace referencia si la fila se actualiza en la tabla primaria.
- SET NULL: Todos los valores que forman la clave externa se establecen en NULL si se actualiza la fila correspondiente de la tabla primaria. Para ejecutar esta restricción, las columnas de clave externa deben admitir valores NULL.
- SET DEFAULT: Todos los valores que forman la clave externa se establecen en los valores predeterminados si se actualiza la fila correspondiente de la tabla primaria. Para ejecutar esta restricción, las columnas de clave externa deben tener valores predeterminados. Si la columna admite valores NULL y no hay ningún valor predeterminado establecido de forma explícita, NULL se convierte en el valor predeterminado implícito de la columna.

### (e) Propiedad IDENTITY

Crea una columna de identidad en una tabla. Esta propiedad se utiliza con las instrucciones CREATE TABLE y ALTER TABLE de Transact-SQL.

Sintaxis

IDENTITY (inicio, incremento)

Siendo inicio el valor que se utiliza para la primera fila cargada en la tabla, e incremento el valor incremental que se agrega al valor de identidad de la anterior fila cargada.

Es necesario especificar la inicialización y el incremento, o no especificar ninguno de los dos. Si no se especifica ninguno, el valor predeterminado es (1,1).

Por ejemplo el código de los empleados (CódigoEm) de la tabla Empleados comenzará por 1 y se incrementará de uno en uno, siendo éste la clave principal.

```
CódigoEm smallint IDENTITY (1, 1) PRIMARY KEY
```

#### Observaciones

- Si hay una columna de identidad para una tabla en la que se realizan eliminaciones frecuentemente, pueden quedar espacios entre los valores de identidad. Si esto constituye un problema, no utilice la propiedad IDENTITY. Sin embargo, para asegurarse de que no se han creado espacios o para rellenar un espacio existente, evalúe los valores de identidad existentes antes de escribir uno explícitamente con SET IDENTITY\_INSERT con valor ON.
- Utilice DBCC CHECKIDENT para comprobar el valor de identidad actual y compararlo con el valor máximo de la columna de identidad.
- Cuando se utiliza la propiedad IDENTITY con CREATE TABLE, SQL Server utiliza la opción NOT FOR REPLICATION de CREATE TABLE para suplantar el incremento automático de una columna de identidad. Normalmente, SQL Server asigna a cada nueva fila insertada en una tabla un valor que contiene un incremento mayor que el valor anterior más alto. Sin embargo, si las nuevas filas están duplicadas desde otro origen de datos, los valores de identidad deben permanecer exactamente como eran en el origen de datos.

### (3) Valores por defecto

Cada columna en un registro debe contener un valor (aún si ese valor es un valor nulo). Hay situaciones en la cuales se necesita cargar una fila de datos en una tabla donde no se conocen los valores para todas las columnas (o esos valores no existen aún). Si la columna permite valores nulos, se puede cargar un valor nulo para esa fila. Dado que como vimos los valores nulos no son aconsejables, una mejor solución puede ser definir un valor por defecto para esa columna. Por ejemplo, es común asignar un valor cero como el valor por defecto (DEFAULT) para columnas numéricas o '' para columnas de caracteres cuando no se especifican valores.

La cláusula DEFAULT en el comando CREATE TABLE se considera una restricción aún cuando en realidad no fuerza a nada.

Cuando se carga una fila en una tabla con una definición de un valor por defecto para una columna, se le está indicando en forma implícita al SQL Server que cargue el valor por defecto en la columna en aquellos casos que no se indique valor para dicha columna.

Si una columna no permite valores nulos y no tiene una definición por defecto, se deberá explícita indicar un valor para esa columna o el SQL Server generará un mensaje de error indicando que la columna no permite valores nulos.

Se puede crear una definición de valores por defecto para una columna de dos maneras:

- Creando la definición del valor por defecto cuando se crea la tabla (como parte de la definición de la tabla)
- Agregando el valor por defecto a una tabla existente (cada columna permite solo un valor por defecto)

El siguiente ejemplo introduce 'Granada' en el campo LocalidadDe para aquellos Departamentos en los que no especifiquemos nada.

*En la creación:*

```
LocalidadDe varchar (50) DEFAULT 'Granada',
```

*Modificando la tabla ya creada*

```
ALTER TABLE Departamentos ADD  
DEFAULT 'Granada' FOR LocalidadDe
```

#### **(4) Crear una tabla a partir de otra**

Este apartado lo voy a explicar mediante el siguiente ejemplo:

Tenemos en la base de datos de Proyectalia una tabla llamada Empleados y queremos crear una tabla llamada NombresEmpleados que contenga de forma única los nombres de todos los empleados.

Vamos a suponer que la tabla NombresEmpleados, no existe, y contendrá la estructura misma que pusimos en la tabla de Empleados, para ello creamos la tabla con los campos necesarios:

```
SELECT DISTINCT NombreEm  
INTO NombreEmpleado  
FROM Empleados
```

La tabla NombreEmpleado se crearía con el campo llamado NombreEm, conteniendo todos los nombres de los distintos empleados.

## Crear una tabla temporal

Estas son tablas similares a las permanentes que se graban en tempdb, y son eliminadas automáticamente cuando ya no son usadas.

Hay dos tipos de tablas temporales, locales y globales, difieren una de la otra en sus nombres, su visibilidad y su ámbito de vida.

- **Tablas Temporales Locales.** El primer carácter del nombre de #, su visibilidad es solamente para la conexión actual del usuario y son eliminadas cuando el usuario se desconecta.
- **Tablas Temporales Globales.** Su nombre comienza con ##, su visibilidad es para cualquier usuario, y son eliminadas luego que todos los usuarios que la referencian se desconectan del SQL Server.

Las tablas temporales se quitan automáticamente cuando están fuera de ámbito, a menos que ya se hayan quitado explícitamente mediante DROP TABLE.

- Una tabla temporal local creada en un procedimiento almacenado se quita automáticamente cuando se completa el procedimiento almacenado. Cualquiera de los procedimientos almacenados anidados ejecutados por el procedimiento almacenado que creó la tabla puede hacer referencia a la tabla. El proceso que llamó al procedimiento almacenado que creó la tabla no puede hacer referencia a la tabla.
- Las demás tablas temporales se quitan automáticamente al final de la sesión actual.
- Las tablas temporales globales se quitan automáticamente cuando la sesión que creó la tabla finaliza y las tareas restantes han dejado de hacer referencia a ellas. La asociación entre una tarea y una tabla se mantiene sólo durante la vida de una única instrucción Transact-SQL. Esto significa que la tabla temporal global se quita al finalizar la última instrucción Transact-SQL que estuviera haciendo referencia activamente a la tabla cuando finalizó la sesión que la creó.

Por ejemplo vamos a crear una tabla temporal llamada 'Temporal1' sobre la tabla alquiler, con los datos IdEjemplar e IdPelícula para las películas que no estén devueltas. Dicha tabla temporal tendrá un carácter local.

```
SELECT IdEjemplar, IdPelícula INTO #Temporal1
FROM Alquiler
WHERE FechaDev IS NULL
```

Mostramos el contenido de la tabla temporal llamada 'temporal':

```
SELECT * FROM #Temporal1
```

Posteriormente la vamos a borrar:

```
DROP TABLE #Temporal1
```

Por ejemplo vamos a crear una tabla temporal llamada 'Temporal2' con los campos IdEjemplar e IdPelícula, ambos de tipo smallint, definiendo la como clave principal la unión de los dos. Dicha tabla temporal tendrá un carácter local.

```
CREATE TABLE #Temporal2 (  
    IdEjemplar smallint,  
    IdPelícula smallint,  
    PRIMARY KEY (IdEjemplar, IdPelícula)  
)
```

Insertamos un registro en la tabla con los valores 4 y 1.

```
INSERT INTO #Temporal2 VALUES (4,1)
```

Mostramos el contenido

```
SELECT * FROM #Temporal2
```

Posteriormente la vamos a borrar:

```
DROP TABLE #Temporal2
```

### (5) Orden de creación atendiendo a las claves ajenas

A la hora de crear las tablas de una base de datos hay que tener en cuenta las claves ajenas, porque si se crea una clave FORANEAS que haga referencia a un atributo de una tabla que todavía no se ha creado, el sistema dará un error ya que no podrá encontrar el atributo referenciado por lo que habrá que crear primero las referenciadas.

También podemos crear todas las tablas y al final mediante modificaciones a la base de datos con ALTER podemos definir todas las claves FORANEAS sin ningún problema.

### iii) Vistas

Una vista es una consulta, que refleja el contenido de una o más tablas, desde la que se puede acceder a los datos como si fuera una tabla.

Dos son las principales razones por las que podemos crear vistas.

- Seguridad, nos pueden interesar que los usuarios tengan acceso a una parte de la información que hay en una tabla, pero no a toda la tabla.
- Comodidad, como hemos dicho el modelo relacional no es el más cómodo para visualizar los datos, lo que nos puede llevar a tener que escribir complejas sentencias SQL, tener una vista nos simplifica esta tarea.

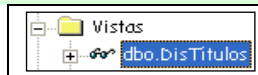
Las vistas no tienen una copia física de los datos, son consultas a los datos que hay en las tablas, por lo que si actualizamos los datos de una vista, estamos actualizando realmente la tabla, y si actualizamos la tabla estos cambios serán visibles desde la vista.

Para crear una vista debemos utilizar la sentencia `CREATE VIEW`, debiendo proporcionar un nombre a la vista y una sentencia SQL `SELECT` válida.

```
CREATE VIEW nombre_vista AS (sentencia_SELECT)
```

Este ejemplo crea una vista sobre la tabla películas, en la que se nos muestren los distintos títulos que tenemos en nuestro VideoClub, se haría de la siguiente forma.

```
CREATE VIEW DisTítulos AS (  
    SELECT DISTINCT Título  
    FROM Películas  
)
```



La vista se puede usar, por ejemplo para mostrar los títulos de las películas anteriores, podríamos escribir:

```
SELECT *  
FROM DisTítulos
```

	Título
1	La bala que dobló la esquina
2	Los otros
3	Mujeres al borde de un ataque de nervios
4	CoCon
5	Ni quito ni pongo

#### iv) Índices

Un índice (INDEX) es una estructura que proporciona un acceso rápido a las filas de una tabla en base a valores de una o más columnas, un propósito similar se puede alcanzar en la creación de la tabla usando la restricción `UNIQUE`. La creación de los índices se debe hacer junto a la creación de la estructura de las tablas, y así, evitar posibles colisiones al crear índices y que la tabla tenga datos, pues si creamos un índice único por campo, y éste tiene valores no únicos, entonces la generación del índice daría un error.

Su formato es el siguiente:

```
CREATE [UNIQUE] INDEX NombreIndice  
ON NombreTabla (Columna(s))
```



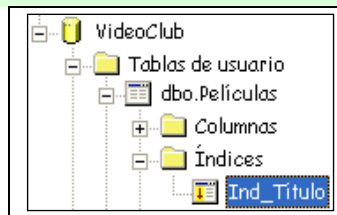
El índice siempre se crea por defecto de menor a mayor, y en columna(s) se pueden especificar más de una columna separadas por comas. Si además de único queremos que el índice no admita valores nulos se pondrá lo siguiente:

```
CREATE UNIQUE INDEX NombreIndice
ON NombreTabla (Columna(s))
WITH IGNORE_DUP_KEY
```

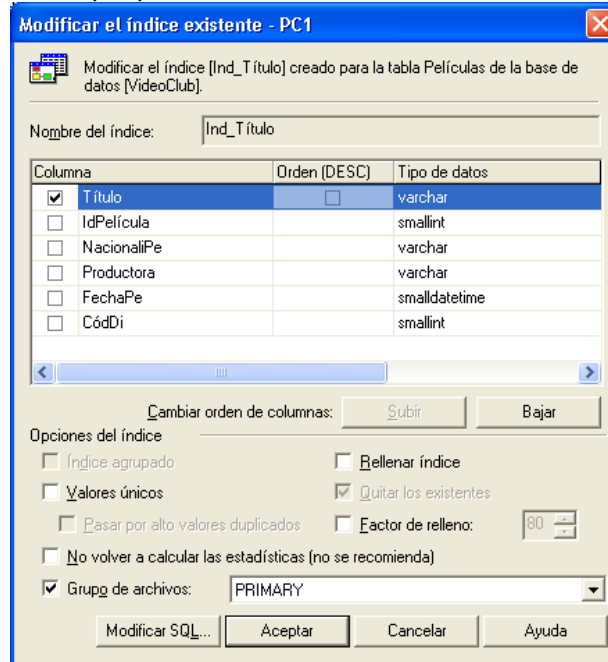
Es una decisión nuestra el establecer los índices oportunos para que el rendimiento de la base de datos no se vea afectado, pues el uso de índices ralentiza la manipulación de los datos.

Por ejemplo vamos a crear un índice sobre la tabla películas para el campo Título, comprobando que no exista, y si así fuese borrándolo.

```
-- Se desactivan los mensajes de salida
SET NOCOUNT OFF
-- si existe el índice lo borra
IF EXISTS (
    SELECT name
      FROM sysindexes
     WHERE name = 'Ind_Título'
)
    DROP INDEX Películas.Título_ind      -- El índice se menciona con tabla.atributo
GO
-- Crea el índice
CREATE INDEX Ind_Título                  -- Creación de índice
    ON Películas (Título)                -- Sobre Películas con el atributo Título
GO
-- Se activan los mensajes de salida
SET NOCOUNT ON
```

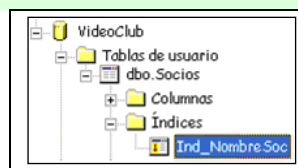


Si editamos las propiedades de este índice veremos lo siguiente:



Ahora vamos a crear un índice sobre la tabla socios para el campo Título, comprobando que no exista, y si así fuese borrándolo.

```
-- si existe el índice lo borra
IF EXISTS (
    SELECT name
    FROM sysindexes
    WHERE name = 'Ind_NombreSoc'
)
DROP INDEX Socios.Ind_NombreSoc
GO
-- Crea el índice
CREATE INDEX Ind_NombreSoc
ON Socios (NombreSoc)
GO
```



Ahora vamos a crear un índice sobre la tabla directores para el campo NombreDi.

```
CREATE INDEX Ind_NombreDi
ON Directores (NombreDi)
```

Igualmente sobre la tabla actores para el campo NombreAc.

```
CREATE INDEX Ind_NombreAc
ON Actores (NombreAc)
```

Posteriormente borramos el índice recién creado:

```
DROP INDEX Actores.Ind_NombreAc
```

Otra solución podría haber sido crear el índice en el momento de la creación de la tabla mediante el siguiente código:

```
NombreAc varchar (50) UNIQUE NOT NULL
```

## b) ALTER

Modifica una definición de una base de datos o una tabla al alterar, agregar o quitar columnas y restricciones, al reasignar particiones o al deshabilitar o habilitar restricciones y desencadenadores.

### i) Base de datos

La sentencia ALTER DATABASE permite realizar cambios en los atributos globales de una base de datos.

Su formato es el siguiente:

```
ALTER DATABASE NombreBaseDatos  
[CHARACTER SET ConjuntoCaracteres]  
[COLLATE ColeccionCaracteres]
```

Por ejemplo con alter le cambiamos el Nombre a la base de datos Proyectalia por EstudioProyectos

```
USE master  
GO  
  
ALTER DATABASE Proyectalia  
    MODIFY NAME = EstudioProyectos  
GO
```

### ii) Tablas

La sentencia ALTER TABLE permite realizar lo siguiente:

- Añadir y eliminar una o varias columnas a una tabla.
- Cambiar el valor por defecto de una columna.
- Añadir o eliminar claves primarias, claves externas, restricciones UNIQUE y restricciones CHECK, ...

Su formato es el siguiente:

```
ALTER TABLE tabla  
    ADD [COLUMN] columna tipo [restricción]  
    ADD clave primaria  
    ADD restricción UNIQUE  
    ADD restricción CHECK  
    DROP [COLUMN] columna
```

## (1) Añadir columnas

La cláusula `ADD COLUMN` (la palabra `COLUMN` es conveniente no ponerlo) permite añadir una columna nueva a la tabla. Como en la creación de tabla, hay que definir la columna indicando su nombre, tipo de datos que puede contener, y si lo queremos alguna restricción de valor no nulo, clave primaria, clave foránea, e índice único, restricción es opcional e indica una restricción que afecta a la columna que estamos definiendo.

Por ejemplo deberemos introducir estas sentencias para añadir un campo llamado `UbicaciónDpto` a la tabla `Departamentos` de la base de datos `Proyectalia`:

```
ALTER TABLE Departamentos  
ADD UbicaciónDpto smallint
```

Otro ejemplo sería el que elimina el campo anteriormente creado.

```
ALTER TABLE Departamentos  
DROP COLUMN UbicaciónDpto
```

## (2) Modificar columnas

Especifica que la columna dada va a cambiarse o modificarse.

Algunos cambios del tipo de datos podrían tener como resultado un cambio en los datos. Por ejemplo, cambiar una columna `nchar` o `nvarchar` por una columna `char` o `varchar` puede dar como resultado la conversión de caracteres extendidos.

Para modificar el tipo de datos almacenado en el campo `UbicaciónDpto` de la tabla `Departamentos` de la base de datos `Proyectalia` deberemos introducir las siguientes sentencias:

```
ALTER TABLE Departamentos  
ALTER COLUMN UbicaciónDpto varchar (10)
```

La columna alterada:

- No puede ser una columna con un tipo de datos `text`, `image`, `ntext` o `timestamp`.
- No puede ser la columna `ROWGUIDCOL` de la tabla.
- No puede ser una columna calculada o utilizarse en una columna calculada.
- Es una columna duplicada.
- No puede utilizarse en un índice, a menos que la columna sea del tipo de datos `varchar`, `nvarchar`, o `varbinary`, el tipo de datos no se cambie y el nuevo tamaño sea igual al tamaño anterior o mayor que éste.
- No puede utilizarse en las estadísticas generadas por la instrucción `CREATE STATISTICS`. Quite primero las estadísticas con la instrucción `DROP STATISTICS`. Las estadísticas generadas automáticamente por el optimizador de consultas se quitan automáticamente con `ALTER COLUMN`.

- No puede utilizarse en una restricción PRIMARY KEY o [FOREIGN KEY] REFERENCES.
- No puede utilizarse en una restricción CHECK o UNIQUE, excepto si se permite alterar la longitud de una columna de longitud variable empleada en una restricción CHECK o UNIQUE.
- No puede estar asociada a un valor predeterminado, excepto que se permita cambiar la longitud, precisión o escala de una columna si no se cambia el tipo de datos.

### (3) Borrar columnas

Para borrar una columna basta con utilizar la cláusula DROP COLUMN (COLUMN es opcional) y el nombre de la columna que queremos borrar, se perderán todos los datos almacenados en la columna.

Por ejemplo para borrar de la base de datos Proyectalia, la columna UbicaciónDpto de la tabla Departamentos deberemos introducir estas sentencias:

```
ALTER TABLE Departamentos  
DROP COLUMN UbicaciónDpto
```

### (4) Modificar restricciones

Se pueden deshabilitar restricciones CHECK preexistentes al ejecutar los comandos INSERT y UPDATE, entonces deshabilitaremos la restricción CHECK durante un comando INSERT o UPDATE si el dato nuevo violara la restricción o si la restricción se debería de aplicar sólo a datos ya existentes en la tabla. Deshabilitar restricciones permite que los datos sean modificados sin que sean validados por las restricciones.

Sabemos que si agregamos una restricción a una tabla que contiene datos, SQL Server los controla para asegurarse que cumplen con la restricción, entonces es posible deshabilitar esta comprobación. Podemos hacerlo al momento de agregar la restricción a una tabla con datos, incluyendo la opción "NOCHECK" en la instrucción "ALTER TABLE"; si se emplea esta opción, los datos no van a cumplir la restricción.

Se pueden deshabilitar las restricciones "CHECK" y "FOREIGN KEY". La opción "opciondechequeo" especifica si se controlan los datos existentes o no con "CHECK" y "NOCHECK" respectivamente. Por defecto, si no se especifica, la opción es "CHECK".

La sintaxis general es:

```
ALTER TABLE NombreTabla  
OpcionDechequeo CONSTRAINT NombreRestriccion
```

Para deshabilitar la restricción de clave foránea que tiene la tabla de Departamentos tendríamos que hacer lo siguiente:

```
ALTER TABLE Departamentos  
NOCHECK CONSTRAINT FK_CódEmDep
```

Para habilitar una restricción deshabilitada, se ejecuta la misma instrucción pero con la cláusula "CHECK" o "CHECK ALL", sobre el ejemplo anterior tendríamos que:

```
ALTER TABLE Departamentos  
CHECK CONSTRAINT FK_CódEmDep
```

Si se emplea "CHECK CONSTRAINT ALL" no se coloca nombre de restricciones, habilita todas las restricciones que tiene la tabla nombrada ("CHECK" y "FOREIGN KEY").

Para saber si una restricción está habilitada o no, podemos ejecutar el procedimiento almacenado "SP\_HELPCONSTRAINT" y entenderemos lo que informa la columna "STATUS\_ENABLED".

Entonces, las cláusulas "CHECK" y "NOCHECK" permiten habilitar o deshabilitar restricciones "FOREIGN KEY" y "CHECK".

Pueden emplearse para evitar la comprobación de datos existentes al crear la restricción o para deshabilitar la comprobación de datos al ingresar, actualizar y eliminar algún registro que infrinja la restricción.

### iii) Vistas

Modifica una vista creada anteriormente. Esto incluye una vista indizada. ALTER VIEW no afecta a desencadenadores ni procedimientos almacenados dependientes y no cambia permisos.

La sintaxis es la siguiente:

```
ALTER VIEW NombreVista VistaCambiar ( Columa [ ,...n ] )  
[WITH AtributoVista [ ,...n ]  
AS Sentencia SELECT  
WITH CHECK OPCIÓN
```

Los permisos de columna se mantienen sólo cuando las columnas tienen los mismos nombres antes y después de que se ejecute ALTER VIEW.

En las columnas de la vista, los permisos de un nombre de columna se aplican a través de una instrucción CREATE VIEW o ALTER VIEW, independientemente del origen de los datos subyacentes.

Si una vista que está actualmente en uso se modifica mediante ALTER VIEW, el Motor de base de datos impone un bloqueo exclusivo de esquema sobre la vista. Cuando se concede el bloqueo, y no hay usuarios activos de la vista, el Motor de base de datos elimina todas las copias de la vista de la caché de procedimientos. Los planes existentes que hacen referencia a la vista permanecen en la caché, pero se vuelven a compilar cuando se llaman.

ALTER VIEW se puede aplicar a vistas indexadas; no obstante, quita incondicionalmente todos los índices de la vista.

#### iv) Índices

Antes de crear una vista, considere estas indicaciones:

- Sólo puede crear vistas en la base de datos actual. Sin embargo, las tablas y las vistas a las que se haga referencia desde la nueva vista pueden encontrarse en otras bases de datos e, incluso, en otros servidores, si la vista se define mediante consultas distribuidas.
- Los nombres de las vistas deben seguir las reglas para los identificadores y ser únicos para cada usuario. Además, el nombre debe ser distinto del de las tablas que posee el usuario.
- Se pueden generar vistas dentro de otras vistas y en procedimientos que hagan referencia a vistas. Microsoft® SQL Server™ 2005 permite anidar hasta 32 niveles de vistas.
- No puede asociar con las vistas reglas ni definiciones DEFAULT.
- Los desencadenadores AFTER no se pueden asociar con las vistas; sólo se pueden asociar los desencadenadores INSTEAD OF.
- La consulta que defina la vista no puede incluir las cláusulas ORDER BY, COMPUTE o COMPUTE BY, ni la palabra clave INTO.
- No se pueden definir definiciones de índice de texto en las vistas.
- No se pueden crear vistas temporales, ni vistas dentro de tablas temporales.
- Las vistas o las tablas que participan en una vista creada con la cláusula SCHEMABINDING no se pueden quitar a menos que se quite o cambie esa vista de forma que deje de tener un enlace de esquema. Además, las instrucciones ALTER TABLE sobre tablas que participan en vistas que tienen enlaces de esquemas provocarán un error si estas instrucciones afectan a la definición de la vista.
- No puede emitir consultas de texto en una vista, aunque una definición de vista puede incluir una consulta de texto si ésta hace referencia a una tabla configurada para la indización de texto.
- Debe especificar el nombre de todas las columnas de la vista en el caso de que:
  - Alguna de las columnas de la vista derive de una expresión aritmética, una función integrada o una constante.
  - Dos o más columnas de la vista tuviesen, en caso contrario, el mismo nombre (normalmente, debido a que la definición de la vista incluye una combinación y las columnas de dos o más tablas diferentes tienen el mismo nombre).

Desea darle a una columna de la vista un nombre distinto del de la columna de la que deriva. (También puede cambiar el nombre de las columnas en la vista). Una columna de una vista hereda los tipos de datos de la columna de la que deriva, aunque no cambie su nombre.

**c) DROP**

Quita una o varias definiciones de bases de datos, tablas y todos los datos, índices, desencadenadores, restricciones y especificaciones de permisos de esas tablas. Las vistas o procedimientos almacenados que hagan referencia a la tabla quitada se deben quitar explícitamente con DROP VIEW o DROP PROCEDURE.

**i) Base de datos**

DROP DATABASE quita las bases de datos dañadas marcadas como sospechosas y quita la base de datos especificada. Antes de quitar una base de datos utilizada en la duplicación, quite antes la duplicación.

La Sintaxis es:

```
DROP DATABASE database_name [ ,...n ]
```

Para borrar la base de datos de Proyectalia deberíamos introducir las siguientes instrucciones:

```
USE master
GO
DROP DATABASE Proyectalia
```

**ii) Tablas**

Quita una definición de tabla y todos los datos, índices, desencadenadores, restricciones y especificaciones de permisos de la tabla.

No se puede utilizar DROP TABLE para quitar una tabla a la que se haga referencia con una restricción FOREIGN KEY. Primero se debe quitar la restricción FOREIGN KEY o la tabla de referencia.

La Sintaxis es:

```
DROP TABLE table_name
```

Para borrar la tabla NombreEmpleado de la base de datos Proyectalia deberíamos introducir las siguientes instrucciones:

```
USE master
GO
DROP TABLE NombreEmpleado
```

**(1) Eliminar columnas**

Para borrar una columna basta con utilizar la cláusula DROP COLUMN (COLUMN es opcional) y el nombre de la columna que queremos borrar, se perderán todos los datos almacenados en la columna.

Por ejemplo para borrar de la base de datos Proyectalia, la columna UbicaciónDpto de la tabla Departamentos deberemos introducir estas sentencias:

```
ALTER TABLE Departamentos
DROP COLUMN UbicaciónDpto
```



## (2) Eliminar restricciones

Para borrar una restricción basta con utilizar la cláusula **DROP CONSTRAINT** y el nombre de la restricción que queremos eliminar, perdiendo así dicha restricción creada con anterioridad.

Para eliminar una restricción (que primero vamos a crear) sobre la tabla **Proyectos** para la columna **CódigoEm** en la base de datos **Proyectalia**.

```
-- Creo la restricción
ALTER TABLE Proyecto ADD
    CONSTRAINT FK_Prueba
        FOREIGN KEY (CódigoEm)
            REFERENCES Empleados (CódigoEm)

-- Elimino la restricción
ALTER TABLE Proyecto
    DROP CONSTRAINT FK_Prueba
```

### iii) Vistas

Quita una o más vistas de la base de datos actual. **DROP VIEW** se puede ejecutar en vistas indizadas.

La Sintaxis es:

```
DROP VIEW { Vista } [ ,...n ]
```

Por ejemplo para borrar la vista que creamos llamada **DisTítulos** de la base de datos de **VideoClub**, usaríamos:

```
DROP VIEW DisTítulos
```

### iv) Índices

Quita uno o más índices de la base de datos actual.

La instrucción **DROP INDEX** no se aplica a los índices creados mediante la definición de las restricciones **PRIMARY KEY** o **UNIQUE** (creados con las opciones **PRIMARY KEY** o **UNIQUE** de las instrucciones **CREATE TABLE** o **ALTER TABLE**, respectivamente).

La Sintaxis es:

```
DROP INDEX 'Tabla.index | Vista.index' [ ,...n ]
```

Por ejemplo para borrar el índice que creamos llamado **Ind\_NombreDi** de la base de datos de **VideoClub** sobre la tabla **Directores**

```
DROP INDEX Directores.Ind_NombreDi
```

## 6) Creación, modificación y eliminación de datos

### a) INSERT

Para almacenar datos en una base de datos debemos insertar filas en las tablas. Para ellos SQL pone a nuestra disposición la sentencia INSERT. Con esta sentencia podemos añadir información a la base de datos. Recordemos que estamos en el modelo relacional, por lo que la información se añadirá a una tabla en forma de filas. Si sólo queremos insertar un valor para un atributo, el resto de los de la tabla deberá contener el valor nulo (NULL). Sin embargo, habrá ciertas ocasiones en que esto no será posible, cuando el atributo esté definido como NO NULO, en cuyo caso deberemos especificar un valor para éste. La sintaxis de esta sentencia es:

```
INSERT INTO tabla (Atributos)
VALUES (Valores)
```

Donde tabla especifica la tabla en la cual se añadirá la fila, atributos es una lista de atributos separados por comas que determinan los atributos para los cuales se darán valores, y valores, especifica los valores que se darán para estos atributos, separados por comas.

### i) Inserción de filas

El proceso de inserción de filas consiste en añadir a una tabla una o más filas y en cada fila todos o parte de sus campos.

Podemos distinguir dos formas de insertar filas:

- Inserción individual de filas.
- Inserción múltiple de filas.

La sintaxis de la sentencia INSERT es diferente según cual sea nuestro propósito.

Sólo podremos omitir un campo al efectuar una inserción cuando este acepte valores nulos.

### ii) Inserción individual de filas

Para realizar la inserción individual de filas SQL posee la instrucción INSERT INTO. La inserción individual de filas es la que más comúnmente utilizaremos. Su sintaxis es la siguiente:

```
INSERT INTO Tabla (Atributos)
VALUES (Valores)
```

Como se puede observar la sentencia tiene dos partes claramente diferenciadas, por un lado la propia INSERT INTO seguida de la lista de atributos en los que queremos insertar los datos, y por otro la lista de valores que queremos insertar en los campos.

Mediante la sentencia INSERT creamos un registro en la tabla Proyectos de la base de datos Proyectalia con los valores especificados, es decir, un 1 para el Código del proyecto, tendría un presupuesto de €100, la función sería "Proyecto número 1" y el Código del empleado el 1.

```
INSERT INTO Proyecto (CódigoPr, PresupPr, Función, CódigoEm)
VALUES (1, €100, 'Proyecto número 1', 1)
```

¿Que ocurriría si ya existiera un proyecto con la CódigoPr 1? Se producirá un error, porque hemos definido la clave primaria en el campo CódigoPr, y como hemos visto la clave primaria debe ser única.

Si omitimos algún par "campo-valor" en la sentencia INSERT, pueden ocurrir varias cosas:

- Que se produzca un error, si el campo no acepta valores nulos.
- Que se grave el registro y se deje nulo el campo, cuando el campo acepte valores nulos.
- Que se grave el registro y se tome el valor por defecto, cuando el campo tenga definido un valor por defecto.

Que hacer en cada momento dependerá del programa.

### iii) Inserción múltiple de filas

La sentencia INSERT permite también insertar varios registros en una tabla. Para ello se utiliza una combinación de la sentencia INSERT junto a una sentencia SELECT. El resultado es que se insertan todos los registros devueltos por la consulta.

```
INSERT INTO <NombreTabla>
[(<Campo1>[,<Campo2>,...])]
SELECT [(<Campo1>[,<Campo2>,...])]
FROM <NombreTablaOrigen>
```

Para poder utilizar la inserción múltiple de filas se deben cumplir las siguientes normas:

- La lista de campos de las sentencias INSERT y SELECT deben coincidir en número y tipo de datos.
- Ninguna de las filas devueltas por la consulta debe infringir las reglas de integridad de la tabla en la que vayamos a realizar la inserción.

Pongamos un ejemplo, sobre la base de datos Proyectalia, insertamos los nombres de los empleados (NombresEmpleados) en la tabla que tenemos para tal fin llamada NombreEmpleado:

```
INSERT INTO NombreEmpleado (NombresEmpleados)
SELECT DISTINCT NombreEm
FROM Empleados
```

**iv) Inserción de valores por posición.**

Podemos omitir los nombres de las columnas siempre que los valores se introduzcan en el mismo orden en que están las columnas creadas en la tabla correspondiente.

Por ejemplo, la siguiente sentencia insertará un registro en la tabla Familiares ya que el orden es DNIFa, NúmeroFa y CódigoEm con los valores correspondientes '10000000A', 3 y 1.

```
INSERT INTO Familiares  
VALUES ('10000000A', 3, 1)
```

**v) Inserción de valores por nombre de columna.**

Podemos especificar el contenido de cada pareja "campo-valor" en el orden que nosotros necesitemos, para ello debemos introducir la lista de los campos en el orden que deseemos y sus correspondientes valores, los demás valores se insertan como NULL o con el valor por defecto (DEFAULT) que le hayamos puesto al crear la tabla o vista.

Por ejemplo, la siguiente sentencia insertará un registro en la tabla Familiares ya que el orden es NúmeroFa, DNIFa y CódigoEm con los valores correspondientes 3, '10000000A' y 1.

```
INSERT INTO Familiares (NúmeroFa, DNIFa, CódigoEm)  
VALUES (3, '10000000A', 1)
```

**vi) Valores por defecto y valores nulos.**

Si al insertar registros no se especifica un valor para un campo que admite valores nulos, se ingresa automáticamente "NULL" y si el campo está declarado "IDENTITY", se inserta el siguiente de la secuencia. A estos valores se les denomina valores por defecto o predeterminados.

Un valor por defecto se inserta cuando no está presente al ingresar un registro y en algunos casos en que el dato ingresado es inválido.

Para campos de cualquier tipo no declarados "NOT NULL", es decir, que admiten valores nulos, el valor por defecto es "NULL". Para campos declarados "NOT NULL", no existe valor por defecto, a menos que se declare explícitamente con la cláusula "DEFAULT".

Para todos los tipos, excepto los declarados "IDENTITY", se pueden explicitar valores por defecto con la cláusula "DEFAULT".

Podemos establecer valores por defecto para los campos cuando creamos la tabla. Para ello utilizamos "DEFAULT" al definir el campo.

También se puede utilizar "DEFAULT" para dar el valor por defecto a los campos en sentencias "INSERT",

Entonces, la cláusula "DEFAULT" permite especificar el valor por defecto de un campo. Si no se explicita, el valor por defecto es "NULL", siempre que el campo no haya sido declarado "NOT NULL".

Los campos para los cuales no se ingresan valores en un "INSERT" tomarán los valores por defecto:

- Si tiene el atributo "IDENTITY": el valor de inicio de la secuencia si es el primero o el siguiente valor de la secuencia, no admite cláusula "DEFAULT";
- Si permite valores nulos y no tiene cláusula "DEFAULT", almacenará "NULL";
- Si está declarado explícitamente "NOT NULL", no tiene valor "DEFAULT" y no tiene el atributo "IDENTITY", no hay valor por defecto, así que causará un error y el "INSERT" no se ejecutará.
- Si tiene cláusula "DEFAULT" (admita o no valores nulos), el valor definido como predeterminado;
- Para campos de tipo fecha y hora, si omitimos la parte de la fecha, el valor predeterminado para la fecha es "01-01-1900", y si omitimos la parte de la hora, "00:00:00".

Un campo sólo puede tener un valor por defecto. Una tabla puede tener todos sus campos con valores por defecto. Que un campo tenga valor por defecto no significa que no admita valores nulos, puede o no admitirlos.

Como ejemplo de estos tres puntos podríamos proponer la siguiente sentencia que creará un registro en la tabla Empleados para el campo CódigoEm se introduciría un valor incremental ya que está definido con IDENTITY (1, 1), para el Jefe se quedará a nulo y para CódigoDe pondría un 1 ya que éste es su valor por defecto, el resto de valores serán las parejas "campo-valor" asignados.

```
INSERT INTO Empleados (NombreEm, DNIEm, DireccEm, SueldoEm)
VALUES ('Nombre del empleado 1', '1111111A', 'La casa del empleado 1', €100.0)
```

## b) UPDATE

El objetivo de la sentencia UPDATE es actualizar los valores de una o varias filas de una tabla, sin necesidad de borrarla e insertarla de nuevo. La sintaxis es la siguiente:

```
UPDATE <Tabla>
SET <Atributo1> = <Valor1>, <Atributo2> = <Valor2>, ...
WHERE <Condición>
```

Donde tabla especifica la tabla donde se encuentran las filas que queremos actualizar, condición especifica la condición que se debe cumplir para actualizar las filas, y lo que viene a continuación de SET especifica la asignación de los nuevos valores a los atributos. Por lo tanto se actualizarán todas las filas que cumplan la condición especificada.

Esta sentencia es la que permite la actualización de la información almacenada en la base datos. Si la sentencia INSERT se utilizaba para añadir nueva información, la sentencia UPDATE se utiliza para modificar la información existente.

Su sintaxis es la siguiente.

```
UPDATE <Tabla>  
  SET <Atributo> = <Valor>  
  FROM <Tablas>  
  WHERE <Condición>
```

Si se especifica más de una asignación de un valor a un atributo, éstas se deberán separar por comas.

La cláusula FROM se puede omitir, en el caso de sólo se necesite acceder a una tabla, que será la misma que la que se actualice.

Ejemplo:

Si queremos cambiar el valor del campo Jefe que hemos insertado anteriormente, deberemos escribir el siguiente código:

```
UPDATE Empleados  
  SET Jefe = 1  
  WHERE DNIEm = '22222222B'
```

Lo que hacemos con la anterior sentencia es buscar el empleado con el DNIEm asignado (condición WHERE), y a continuación actualizar el valor del atributo Jefe de la fila obtenida a 1. Si ejecutamos la anterior sentencia, obtenemos el resultado: (1 filas afectadas) lo que quiere decir que la fila ha sido actualizada con éxito.

#### i) Cambiar una columna de una fila

Es la forma más sencilla de utilizar la sentencia UPDATE.

Por ejemplo si queremos cambiar el atributo de jefe de la tabla Empleado a 2 para aquellos que el DNI del empleado sea '11111111A'

```
UPDATE Empleados  
  SET Jefe = 2  
  WHERE DNIEm = '11111111A'
```

#### ii) Cambiar varias columnas de una fila

Por ejemplo si queremos cambiar la columna Jefe y CódigoDe de la tabla Empleados a 2 y 1 respectivamente para aquellos que el DNI del empleado sea '11111111A'

```
UPDATE Empleados  
  SET Jefe = 2, CódigoDe = 1  
  WHERE DNIEm = '11111111A'
```

#### iii) Modificar datos en varias filas

Por ejemplo si queremos cambiar los atributos de Jefe y CódigoDe a 4 y 3 respectivamente de la tabla Empleados donde el CódigoEm sea 1,2 ó 3

```
UPDATE Empleados  
  SET Jefe = 4, CódigoDe = 3  
  WHERE CódigoDe IN (1, 2, 3)
```

**iv) Uso de expresiones en la asignación**

Por ejemplo si queremos cambiar la columna del CódigoDe a 1 para los empleados cuyo jefe sea el 1 y 2

```
UPDATE Empleados  
SET CódigoDe = 1  
WHERE (Jefe= 1 And Jefe = 2)
```

**v) Valores por defecto y valores nulos**

Nos sirve para modificar un valor con su valor por defecto. Se debe especificar a la hora de crear las tablas mediante la opción DEFAULT, o modificando las propiedades con ALTER TABLE.

Para asignar un valor de un atributo como un valor nulo, en la creación de la tabla, tenemos que poner que acepta nulos, de lo contrario daría un error.

Por ejemplo si queremos cambiar el valor por defecto para la columna en el atributo Jefe de la tabla de Empleados de la base de datos Proyectalia.

```
UPDATE Empleados  
SET Jefe = DEFAULT
```

Otro ejemplo sería introducir un valor nulo en el atributo Jefe de la tabla Empleados de la base de datos Proyectalia.

```
UPDATE Empleados  
SET Jefe = NULL
```

**c) DELETE**

Para borrar datos de una tabla, debemos utilizar la sentencia DELETE.

El objeto de la sentencia DELETE es el de borrar filas de una tabla. Para poder borrar filas en una tabla se deben cumplir las condiciones de seguridad determinadas por el administrador (se verán en un próximo capítulo), y deben de cumplirse también las reglas de integridad referencial. La sintaxis es la siguiente:

```
DELETE FROM Tabla  
WHERE Condición
```

Donde tabla especifica la tabla sobre la cual queremos borrar las filas, y condición especifica la condición que se debe cumplir para que se borren las filas. Si omitimos la condición, se borrarán todas las filas de la tabla.

Por ejemplo, las sentencias que aparecen a continuación borrarían todas las filas de la tabla Empleados.

```
DELETE Empleados
```

Por ejemplo, si queremos borrar la fila que hemos creado en la tabla autores, deberemos ejecutar el siguiente código, obteniendo el siguiente resultado: (1 fila afectada)

```
DELETE  
FROM Empleados  
WHERE Jefe = 1
```



Para comprobar que la fila se ha borrado, ejecutamos la sentencia que muestra el siguiente ejemplo, cuyo resultado es: (0 filas afectadas), lo que quiere decir que la fila no se encuentra en la tabla, es decir, ha sido borrada.

```
SELECT *  
FROM Empleados  
WHERE Jefe = 1
```

El siguiente ejemplo ilustra el uso de la sentencia DELETE. Es buena idea especificar en la sentencia WHERE los campos que forman la clave primaria de la tabla para evitar borrar datos que no queramos eliminar.

```
DELETE  
FROM Empleados  
WHERE Jefe = 1
```

Cuando trabajemos con la sentencia DELETE debemos tener en cuenta las siguientes consideraciones:

- Solo podemos borrar datos de una única tabla.
- Cuando borramos datos de una vista, los estamos borrando también de la tabla. Las vistas son solo una forma de ver los datos, no una copia.
- Si intentamos borrar un registro de una tabla referenciada por una FOREIGN KEY como tabla maestra, si la tabla dependiente tiene registros relacionados la sentencia DELETE fallará.

#### d) TRUNCATE

Para realizar un borrado completo de tabla debemos considerar la posibilidad de utilizar la sentencia TRUNCATE, mucho más rápida que DELETE.

La sintaxis de la sentencia TRUNCATE es la siguiente:

```
TRUNCATE TABLE tabla
```

Esta sentencia funciona de la misma forma que la instrucción DELETE sin especificar una cláusula WHERE: ambas quitan todas las filas de la tabla. Pero TRUNCATE TABLE es más rápida y utiliza menos recursos de los registros de transacciones y de sistema que DELETE.

La instrucción DELETE quita una a una las filas y graba una entrada en el registro de transacciones por cada fila eliminada. TRUNCATE TABLE quita los datos al cancelar la asignación de las páginas de datos utilizadas para almacenar los datos de la tabla y sólo se graba en el registro de transacciones la página de asignaciones quitadas.

TRUNCATE TABLE quita todas las filas de una tabla, pero permanece la estructura y sus columnas, restricciones, índices, etc. El contador utilizado por una identidad para las nuevas filas se restablece al valor de inicialización de la columna. Si desea conservar el valor del contador, se debe utilizar DELETE en su lugar.



El siguiente ejemplo muestra el uso de la sentencia TRUNCATE, sobre la tabla empleados de la base de datos Proyectalia.

```
TRUNCATE TABLE Empleados
```

Cuando trabajemos con la sentencia TRUNCATE debemos tener en cuenta las siguientes consideraciones.

- La sentencia TRUNCATE no es transaccional. No se puede deshacer.
- La sentencia TRUNCATE no admite cláusula WHERE. Borra toda la tabla.
- No todos los gestores de bases de datos admiten la sentencia TRUNCATE.

## 7) Consultas: SELECT

La sentencia SELECT es una sentencia SQL, que pertenece al conjunto del Lenguaje de Manipulación de Datos, y que sirve para recuperar registros de una o varias tablas, de una o varias bases de datos. Su sintaxis es la siguiente:

```
SELECT Atributos FROM Tablas
[WHERE Condición]
[GROUP BY Atributos]
[HAVING Condición]
[ORDER BY Atributos]
```

Veamos por partes que quiere decir cada una de las partes que conforman la sentencia.

	Significado
<b>SELECT</b>	Palabra clave que indica que la sentencia de SQL que queremos ejecutar es de selección.
<b>ALL</b>	Indica que queremos seleccionar todos los valores. Es el valor por defecto y no suele especificarse casi nunca.
<b>DISTINCT</b>	Indica que queremos seleccionar sólo los valores distintos.
<b>FROM</b>	Indica la tabla (o tablas) desde la que queremos recuperar los datos. En el caso de que exista más de una tabla se denomina a la consulta "consulta combinada" o "JOIN". En las consultas combinadas es necesario aplicar una condición de combinación a través de una cláusula <b>WHERE</b> .
<b>WHERE</b>	Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Admiten los operadores lógicos <b>AND</b> y <b>OR</b> .
<b>GROUP BY</b>	Especifica la agrupación que se da a los datos. Se usa siempre en combinación con funciones agregadas.
<b>HAVING</b>	Especifica una condición que debe cumplirse para los datos. Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Su funcionamiento es similar al de <b>WHERE</b> pero aplicado al conjunto de resultados devueltos por la consulta. Debe aplicarse siempre junto a <b>GROUP BY</b> y la condición debe estar referida a los campos contenidos en ella.
<b>ORDER BY</b>	Presenta el resultado ordenado por las columnas indicadas. El orden puede expresarse con <b>ASC</b> (orden ascendente) y <b>DESC</b> (orden descendente). El valor predeterminado es <b>ASC</b> .

En el caso de que se especifiquen varias tablas, en la cláusula FROM, será conveniente denotar los campos de la cláusula SELECT precedidos por el nombre de la tabla donde se encuentra y un punto, para que, en el caso de que dicho campo exista en más de una tabla, se sepa en cada momento a cual de ellos nos estamos refiriendo, evitando en este caso el problema de ambigüedad. Por ejemplo podríamos poner Empleados.NombreEm en vez de NombreEm que sería más ambiguo.

### a) Sintaxis de las consultas

```
SELECT Atributos FROM Tablas
[WHERE Condición]
[GROUP BY Atributos]
[HAVING Condición]
[ORDER BY Atributos]
```

## i) La cláusula WHERE

La forma de usar el Transact SQL para realizar consultas realizando una proyección horizontal (es decir, seleccionando las filas), es mediante la opción WHERE. A continuación de esta palabra reservada, se debe especificar la condición lógica que se debe evaluar, para obtener aquellas filas que la cumplen.

Para expresar una expresión lógica se pueden emplear cualquiera de los operadores de Transact SQL cuyo resultado devuelva un valor lógico, aunque también se pueden utilizar operadores de cadena.

Estos operadores son los siguientes:

Operador	Acción															
>	Compara si una expresión es mayor que otra.															
<	Compara si una expresión es menor que otra.															
>=	Compara si una expresión es mayor o igual que otra.															
<=	Compara si una expresión es menor o igual que otra.															
<>	Compara si una expresión es distinta que otra.															
=	<div>Compara si una expresión es igual que otra.</div> <div>Por ejemplo para mostrar los títulos de las películas de nacionalidad 'Española'.</div> <div>SELECT Título, NacionaliPe</div> <div>FROM Películas</div> <div>WHERE NacionaliPe = 'Española'</div> <table><thead><tr><th></th><th>Título</th><th>NacionaliPe</th></tr></thead><tbody><tr><td>1</td><td>La bala que dobló la esquina</td><td>Española</td></tr><tr><td>2</td><td>Los otros</td><td>Española</td></tr><tr><td>3</td><td>Mujeres al borde de un ataque de nervios</td><td>Española</td></tr></tbody></table>		Título	NacionaliPe	1	La bala que dobló la esquina	Española	2	Los otros	Española	3	Mujeres al borde de un ataque de nervios	Española			
	Título	NacionaliPe														
1	La bala que dobló la esquina	Española														
2	Los otros	Española														
3	Mujeres al borde de un ataque de nervios	Española														
LIKE	<div>Compara todas las cadenas que verifican un patrón de búsqueda.</div> <div>Por ejemplo para mostrar los títulos de las películas de nacionalidad comience por 'A'</div> <div>SELECT Título, NacionaliPe</div> <div>FROM Películas</div> <div>WHERE NacionaliPe LIKE('A%')</div> <table><thead><tr><th></th><th>Título</th><th>NacionaliPe</th></tr></thead><tbody><tr><td>1</td><td>CoCon Americana</td><td></td></tr></tbody></table>		Título	NacionaliPe	1	CoCon Americana										
	Título	NacionaliPe														
1	CoCon Americana															
NOT LIKE	<div>Compara todas las cadenas que no verifican un patrón de búsqueda.</div> <div>Por ejemplo para mostrar los títulos de las películas cuya nacionalidad no comience por 'A'</div> <div>SELECT Título, NacionaliPe</div> <div>FROM Películas</div> <div>WHERE NacionaliPe NOT LIKE ('A%')</div> <table><thead><tr><th></th><th>Título</th><th>NacionaliPe</th></tr></thead><tbody><tr><td>1</td><td>La bala que dobló la esquina</td><td>Española</td></tr><tr><td>2</td><td>Los otros</td><td>Española</td></tr><tr><td>3</td><td>Mujeres al borde de un ataque de nervios</td><td>Española</td></tr><tr><td>4</td><td>Ni quito ni pongo</td><td>Japonesa</td></tr></tbody></table>		Título	NacionaliPe	1	La bala que dobló la esquina	Española	2	Los otros	Española	3	Mujeres al borde de un ataque de nervios	Española	4	Ni quito ni pongo	Japonesa
	Título	NacionaliPe														
1	La bala que dobló la esquina	Española														
2	Los otros	Española														
3	Mujeres al borde de un ataque de nervios	Española														
4	Ni quito ni pongo	Japonesa														

Operador	Acción																								
BETWEEN	<p>Compara todas las cadenas que están comprendidas en un rango de valores. Por ejemplo para mostrar los Identificadores y títulos de las películas, cuyo identificador este entre 1 y 3</p> <pre>SELECT IdPelícula, Título       FROM Películas       WHERE IdPelícula BETWEEN 1 AND 3</pre> <table><tr><th></th><th>IdPelícula</th><th>Título</th></tr><tr><td>1</td><td>1</td><td>La bala que dobló la esquina</td></tr><tr><td>2</td><td>2</td><td>Los otros</td></tr><tr><td>3</td><td>3</td><td>Mujeres al borde de un ataque de nervios</td></tr></table>		IdPelícula	Título	1	1	La bala que dobló la esquina	2	2	Los otros	3	3	Mujeres al borde de un ataque de nervios												
	IdPelícula	Título																							
1	1	La bala que dobló la esquina																							
2	2	Los otros																							
3	3	Mujeres al borde de un ataque de nervios																							
IN	<p>Compara todas las cadenas que están contenidas en una lista. Por ejemplo para mostrar los Identificadores y títulos de las películas, cuyo identificador sea 1, 2 o 3</p> <pre>SELECT IdPelícula, Título       FROM Películas       WHERE IdPelícula IN (1, 2, 3)</pre> <table><tr><th></th><th>IdPelícula</th><th>Título</th></tr><tr><td>1</td><td>1</td><td>La bala que dobló la esquina</td></tr><tr><td>2</td><td>2</td><td>Los otros</td></tr><tr><td>3</td><td>3</td><td>Mujeres al borde de un ataque de nervios</td></tr></table>		IdPelícula	Título	1	1	La bala que dobló la esquina	2	2	Los otros	3	3	Mujeres al borde de un ataque de nervios												
	IdPelícula	Título																							
1	1	La bala que dobló la esquina																							
2	2	Los otros																							
3	3	Mujeres al borde de un ataque de nervios																							
IS NULL	<p>Determina si la expresión especificada contiene el valor nulo. Mostrar los códigos de las películas que no estén devueltas.</p> <pre>SELECT IdPelícula       FROM Alquiler       WHERE FechaDev IS NULL</pre> <table><tr><th></th><th>IdPelícula</th></tr><tr><td>1</td><td>1</td></tr><tr><td>2</td><td>4</td></tr><tr><td>3</td><td>1</td></tr><tr><td>4</td><td>2</td></tr></table>		IdPelícula	1	1	2	4	3	1	4	2														
	IdPelícula																								
1	1																								
2	4																								
3	1																								
4	2																								
IS NOT NULL	<p>Determina si la expresión especificada no contiene el valor nulo. Mostrar los códigos de las películas que estén devueltas.</p> <pre>SELECT IdPelícula       FROM Alquiler       WHERE FechaDev IS NOT NULL</pre> <table><tr><th></th><th>IdPelícula</th></tr><tr><td>1</td><td>1</td></tr><tr><td>2</td><td>1</td></tr><tr><td>3</td><td>1</td></tr><tr><td>4</td><td>2</td></tr><tr><td>5</td><td>1</td></tr><tr><td>6</td><td>4</td></tr><tr><td>7</td><td>1</td></tr><tr><td>8</td><td>1</td></tr><tr><td>9</td><td>3</td></tr><tr><td>10</td><td>5</td></tr><tr><td>11</td><td>1</td></tr></table>		IdPelícula	1	1	2	1	3	1	4	2	5	1	6	4	7	1	8	1	9	3	10	5	11	1
	IdPelícula																								
1	1																								
2	1																								
3	1																								
4	2																								
5	1																								
6	4																								
7	1																								
8	1																								
9	3																								
10	5																								
11	1																								

## ii) La cláusula GROUP BY

La cláusula GROUP BY agrupa, como su propio nombre indica, filas que tienen el mismo valor para un atributo, en grupos distintos.

Ejemplos

Mostrar los estados de conservación los ejemplares, indicando el estado y el número de ejemplares de cada estado de conservación.

```
SELECT EstadoCons, Count(*) AS 'Número de ejemplares'
FROM Ejemplares
GROUP BY EstadoCons
```

	EstadoCons	Número de ejemplares
1	Bueno	18
2	Malo	1
3	Regular	3

Mostrar cuántas películas hay de cada nacionalidad.

```
SELECT NacionaliPe, Count(*) AS 'Número de ejemplares'
FROM Películas
GROUP BY NacionaliPe
```

	NacionaliPe	Número de ejemplares
1	Americana	1
2	Española	3
3	Japonesa	1

Mostrar cuántos actores hay de cada sexo

```
SELECT SexoAc, Count(*) AS 'Número de ejemplares'
FROM Actores
GROUP BY SexoAc
```

	SexoAc	Número de ejemplares
1	Hombre	4
2	Mujer	3

## iii) La cláusula HAVING

La cláusula HAVING es similar a la cláusula WHERE, salvo que aquella se usa como condición de búsqueda cuando se especifica la cláusula GROUP BY. Por lo tanto el funcionamiento es similar al ya visto para WHERE. La única diferencia es que HAVING se aplica a condiciones de grupo.

Ejemplos:

Mostrar los estados de conservación de los ejemplares, indicando el estado y el número de ejemplares de cada estado de conservación, pero solamente de los que tenga más de 5 ejemplares.

```
SELECT EstadoCons, Count(*) AS 'Número de ejemplares'
FROM Ejemplares
GROUP BY EstadoCons
HAVING Count(*)>5
```

	EstadoCons	Número de ejemplares
1	Bueno	18

#### iv) La cláusula ORDER BY

La cláusula ORDER BY determina el orden de visualización de las filas obtenidas en la sentencia SELECT. A continuación de dicha palabra reservada, se debe especificar el atributo o los atributos por los cuales se ordenará el resultado obtenido en la consulta.

Con esta cláusula se altera el orden de visualización de las filas de la tabla pero en ningún caso se modifica el orden de las filas dentro de la tabla. La tabla no se modifica.

Por defecto el orden será ascendente (ASC) (de menor a mayor si el campo es numérico, por orden alfabético si el campo es de tipo texto, de anterior a posterior si el campo es de tipo fecha/hora, etc...

Por ejemplo para mostrar los títulos de las películas de nacionalidad 'Española' ordenados por el título.

```
SELECT Título, NacionaliPe
FROM Películas
WHERE NacionaliPe = 'Española'
ORDER BY Título
```

	Título	NacionaliPe
1	La bala que dobló la esquina	Española
2	Los otros	Española
3	Mujeres al borde de un ataque de nervios	Española

Si queremos podemos alterar ese orden utilizando la cláusula DESC (DESCendente), en este caso el orden será el inverso al ASC.

Por ejemplo para mostrar los títulos de las películas de nacionalidad 'Española' ordenados descendentemente por el título.

```
SELECT Título, NacionaliPe
FROM Películas
WHERE NacionaliPe = 'Española'
ORDER BY Título DESC
```

	Título	NacionaliPe
1	Mujeres al borde de un ataque de nervios	Española
2	Los otros	Española
3	La bala que dobló la esquina	Española

También podemos ordenar por varias columnas, en este caso se indican los atributos separados por comas.

Se ordenan las filas por la primera columna de ordenación, para un mismo valor de la primera columna, se ordenan por la segunda columna, y así sucesivamente.

La cláusula DESC o ASC se puede indicar para cada columna y así utilizar una ordenación distinta para cada columna.

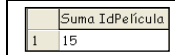
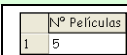

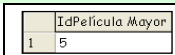
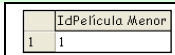
Por ejemplo para mostrar los títulos de las películas ordenados descendientemente por la nacionalidad y el ascendientemente por el título.

```
SELECT Título, NacionaliPe
FROM Películas
ORDER BY NacionaliPe DESC, Título
```

	Título	NacionaliPe
1	Ni quito ni pongo	Japonesa
2	La bala que dobló la esquina	Española
3	Los otros	Española
4	Mujeres al borde de un ataque de nervios	Española
5	CoCon	Americana

### v) Funciones escalares para SELECT

Entendemos por funciones escalares, todas aquellas que permiten realizar operaciones de conteo de filas, suma de atributos, obtención de medias, etc. Dichas funciones se especifican a continuación de la palabra reservada SELECT. Las funciones que soporta la sentencia SELECT en el Transact SQL son las siguientes:

Función	Acción	Ejemplo
<b>Sum</b>	Realiza una suma acumulativa de un atributo para todas las filas accedidas mediante una consulta SQL.	Mostrar la suma de los identificadores de las películas que tenemos: <pre>SELECT Sum(IdPelícula) AS 'Suma IdPelícula' FROM Películas</pre> 
<b>Count</b>	Cuenta todas las filas de las tablas accedidas mediante una consulta SQL.	Mostrar el número de películas que tenemos: <pre>SELECT Count(*) AS 'Nº Películas' FROM Películas</pre> 
<b>Avg</b>	Realiza una media aritmética de los atributos para todas las filas accedidas mediante la consulta SQL.	Mostrar la media de los identificadores de las películas que tenemos: <pre>SELECT Avg(IdPelícula) AS 'Media IdPelícula' FROM Películas</pre> 
<b>Max</b>	Obtiene el máximo valor del atributo especificado, de entre todas las filas seleccionadas mediante la sentencia SQL.	Mostrar el identificador más grande que tenemos: <pre>SELECT Max(IdPelícula) AS 'IdPelícula Mayor' FROM Películas</pre> 
<b>Min</b>	Obtiene el mínimo valor del atributo especificado, de entre todas las filas seleccionadas mediante la sentencia SQL.	Mostrar el identificador más pequeño que tenemos: <pre>SELECT Min(IdPelícula) AS 'IdPelícula Menor' FROM Películas</pre> 

**b) Consultas sencillas sobre una tabla (o una vista)****i) Que muestre todos los atributos y todas las filas (\*)**

Mostrar todas las filas de la tabla películas.

```
SELECT *
FROM Películas
```

	IdPelícula	Título	NacionaliPe	Productora	FechaPe	CódDi
1	1	La bala que dobló la esquina	Española	Hermanos García S.A.	1989-10-23 00:00:00	5
2	2	Los otros	Española	Producciones El aguila real	1956-12-12 00:00:00	2
3	3	Mujeres al borde de un ataque de nervios	Española	Maricomix S.L.	1963-12-05 00:00:00	1
4	4	CoCon	Americana	Columbia pictures	1963-05-08 00:00:00	5
5	5	Ni quito ni pongo	Japonesa	Yujara	1967-11-15 00:00:00	5

**ii) Que muestre todos los atributos y filas seleccionadas (WHERE con operadores de relación, IN, BETWEEN, LIKE, IS NULL, ANY, ALL, EXISTS, ...)**

**IN**, este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los que están en una lista o subconsulta. Su sintaxis es:  
 Expresión [Not] In(Valor1, Valor2, ...)

O bien:

Expresión [Not] In (Subconsulta)

Por ejemplo para mostrar los identificadores y títulos de las películas, cuyo identificador sea 1, 2 ó 3

```
SELECT IdPelícula, Título
FROM Películas
WHERE IdPelícula IN (1, 2, 3)
```

	IdPelícula	Título
1	1	La bala que dobló la esquina
2	2	Los otros
3	3	Mujeres al borde de un ataque de nervios

Por ejemplo para mostrar los identificadores y títulos de las películas, que estén alquiladas y no devueltas.

```
SELECT IdPelícula, Título
FROM Películas
WHERE IdPelícula IN (SELECT DISTINCT IdPelícula
                     FROM Alquila
                     WHERE FechaDev IS NULL)
```

	IdPelícula	Título
1	1	La bala que dobló la esquina
2	2	Los otros
3	4	CoCon



Mostrar todas las filas donde la fecha de alquiler de la tabla Alquiler esté entre el 01/06/2008 y el 31/12/2008 y no están devueltas.

```
SELECT *
FROM Alquiler
WHERE FechaAlqu BETWEEN '01/06/2008' AND '31/12/2008' AND
FechaDev IS NULL
```

	FechaAlqu	IdEjemplar	IdPelícula	IdSoc	FechaDev
1	2008-11-10 00:00:00	3	1	2	NULL
2	2008-11-10 00:00:00	17	4	2	NULL
3	2008-11-24 00:00:00	1	1	1	NULL
4	2008-11-24 00:00:00	5	2	5	NULL

**LIKE** se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

Expresión LIKE Modelo

En donde expresión es una cadena modelo o campo contra el que se compara expresión. Se puede utilizar el operador LIKE para encontrar valores en los campos que coincidan con el modelo especificado. Por modelo puede especificar un valor completo (LIKE 'Ana María'), o se pueden utilizar caracteres comodín para encontrar un rango de valores (LIKE 'An%').

El operador LIKE se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena. Por ejemplo, para mostrar los títulos de las películas que empiecen por 'C'.

```
SELECT Título
FROM Películas
WHERE Título LIKE ('C%')
```

Título
1 CoCon

Mostrar los códigos de las películas que no estén devueltas.

```
SELECT IdPelícula
FROM Alquiler
WHERE FechaDev IS NULL
```

IdPelícula
1 1
2 4
3 1
4 2

**SOME | ANY** se utiliza cuando el elemento que queremos comparar es (igual, distinto, mayor, menor, etc.) que alguno de los elementos que está dentro de ANY(...). Tiene que devolver algún valor del mismo tipo (aunque sea de otra tabla pero los dos del mismo tipo, int, varchar, etc.), y además un único campo.

Por ejemplo para mostrar los Directores que hayan dirigido alguna película.

```
SELECT NombreDi
FROM Directores
WHERE CódDi = ANY (SELECT CódDi FROM Películas)
```

	NombreDi
1	Pedro Almodovar
2	Alejandro Amenabar
3	Mariano la Piedara

**ALL** se utiliza cuando el elemento que queremos comparar es (igual, distinto, mayor, menor, etc.) que todos los elementos que está dentro de ALL(...). Tiene que devolver algún valor del mismo tipo (aunque sea de otra tabla pero los dos del mismo tipo, int, varchar, etc.), y además un único campo.

Por ejemplo para mostrar todos Directores que NO hayan dirigido ninguna película.

```
SELECT CódDi
FROM Directores
WHERE CódDi <> ALL (
    SELECT DISTINCT CódDi
    FROM Películas
)
```

O bien

```
SELECT CódDi
FROM Directores
WHERE NOT EXISTS (
    SELECT CódDi
    FROM Películas
    WHERE Directores.CódDi = CódDi
)
```

	CódDi
1	3
2	4

**EXISTS:** se utiliza con una subconsulta para probar la existencia de filas devueltas por la subconsulta. Comprueba si devuelve algo, da lo mismo que sea un campo o varios, pero que devuelva algo.

Por ejemplo para mostrar los títulos de las películas que tengan al menos un ejemplar en la tabla de ejemplares.

```
SELECT Título
FROM Películas
WHERE EXISTS (SELECT *
    FROM Ejemplares
    WHERE Películas.IdPelícula = Ejemplares.IdPelícula)
```

	Título
1	La bala que dobló la esquina
2	Los otros
3	Mujeres al borde de un ataque de nervios
4	CoCon
5	Ni quito ni pongo

**NOT EXISTS:** se utiliza con una subconsulta para probar la NO existencia de filas devueltas por la subconsulta. Comprueba si no devuelve algo, da lo mismo que sea un campo o varios, pero que devuelva algo.

Por ejemplo para mostrar los títulos de las películas que NO tengan al menos un ejemplar en la tabla de ejemplares.

```
SELECT Título
FROM Películas
WHERE NOT EXISTS (SELECT *
                  FROM Ejemplares
                  WHERE Películas.IdPelícula = Ejemplares.IdPelícula)
```

Título
--------

### iii) Que muestre algunos atributos y todas las filas (- ALL)

Si no se incluye ninguno de los predicados se asume ALL. El Motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL. No es conveniente abusar de este predicado ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados.

Por ejemplo para mostrar los Códigos de todas las películas que se hallan alquilado, aunque estén repetidas.

```
SELECT ALL IdPelícula
FROM Alquila
```

	IdPelícula
1	1
2	4
3	1
4	1
5	1
6	2
7	1
8	4
9	1
10	1
11	1
12	2
13	3
14	5
15	1

Las siguientes instrucciones mostrarían los identificadores de las películas alquiladas de forma única:

```
SELECT DISTINCT IdPelícula
FROM Alquila
```

	IdPelícula
1	1
2	2
3	3
4	4
5	5

Muestra los distintos estados de conservación posibles de los ejemplares (sin repeticiones)

```
SELECT DISTINCT EstadoCons
FROM Ejemplares
```

	EstadoCons
1	Bueno
2	Malo
3	Regular

#### iv) Que muestre algunos atributos y las filas no repetidas (DISTINCT)

Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos.

Por ejemplo para mostrar los códigos de las películas que estén alquiladas pero sin que se repita el código de la película.

```
SELECT DISTINCT IdPelícula
FROM Alquila
WHERE FechaDev IS NULL
```

	IdPelícula
1	1
2	2
3	4

#### v) Que muestre atributos y tablas con alias (AS)

En determinadas circunstancias es necesario asignar un nombre a alguna columna determinada de un conjunto devuelto, otras veces por simple capricho o por otras circunstancias. Para resolver todas ellas tenemos la palabra reservada AS que se encarga de asignar el nombre que deseamos a la columna deseada.

Por ejemplo para mostrar el título de la película y la nacionalidad, utilizando alias para ambos campos.

```
SELECT Título AS 'Título de la Película', NacionaliPE AS Nacionalidad
FROM Películas
```

	Título de la Película	Nacionalidad
1	La bala que dobló la esquina	Española
2	Los otros	Española
3	Mujeres al borde de un ataque de nervios	Española
4	CoCon	Americana
5	Ni quito ni pongo	Japonesa

Otra forma de hacer lo mismo pero indicando un alias para la tabla películas sería:

```
SELECT Título AS 'Título de la Película', NacionaliPE AS Nacionalidad
FROM Películas AS P
```

O bien

```
SELECT Título 'Título de la Película', NacionaliPE Nacionalidad
FROM Películas P
```

**vi) Que muestre algún campo como resultado de la operación de un atributo (columna calculada)**

Muestra el identificador de cada película alquilada, el número de días que estuvo alquilada, y el precio de cada alquiler sabiendo que se paga 0,5€ por día de alquiler.

```
SELECT IdPelícula, DATEDIFF(dd, FechaAlqu, FechaDev) AS Días,
      (DATEDIFF(dd, FechaAlqu, FechaDev)*0.5) AS Importe
FROM Alquila
WHERE FechaDev IS NOT NULL
```

	IdPelícula	Días	Importe
1	1	3	1.5
2	1	3	1.5
3	1	0	.0
4	2	1	.5
5	1	4	2.0
6	4	4	2.0
7	1	2	1.0
8	1	2	1.0
9	3	2	1.0
10	5	2	1.0
11	1	0	.0

Muestra cuantos días ha estado una película alquilada.

```
SELECT IdPelícula, DATEDIFF(dd, FechaAlqu, FechaDev) AS Días
FROM Alquila
WHERE FechaDev IS NOT NULL
```

	IdPelícula	Días
1	1	3
2	1	3
3	1	0
4	2	1
5	1	4
6	4	4
7	1	2
8	1	2
9	3	2
10	5	2
11	1	0

**vii) Que muestre las primeras n filas (TOP, %)**

Devuelve un cierto número de registros que entran entre al principio o al final de un rango especificado por una cláusula ORDER BY.

Por ejemplo para mostrar los nombres de los tres primeros socios.

```
SELECT TOP (3) NombreSoc
FROM Socios
```

	NombreSoc
1	ángela López
2	Antonio Megías
3	Azucena Benítez

Por ejemplo para mostrar los nombres del 25% del número total de socios.

```
SELECT TOP 25 PERCENT NombreSoc  
FROM Socios
```

	NombreSoc
1	ángela López
2	Antonio Megías

**viii) Que muestre las filas ordenadas por un atributo (ORDER BY, ASC, DESC)**

Muestra los nombres de los tres primeros socios después de ordenarlos por orden descendente del nombre.

```
SELECT TOP 3 NombreSoc  
FROM Socios  
ORDER BY NombreSoc DESC
```

	NombreSoc
1	Isaac Torralba
2	Francisco Mariano Prieto
3	Francisco Barragán

Muestra los nombres del 30% de los primeros socios después de ordenarlos por orden descendente del nombre.

```
SELECT TOP 30 PERCENT NombreSoc  
FROM Socios  
ORDER BY NombreSoc DESC
```

	NombreSoc
1	Isaac Torralba
2	Francisco Mariano Prieto
3	Francisco Barragán

Muestra el código del socio con los alquileres de películas no devueltas poniendo primero los alquileres más antiguos.

```
SELECT IdSoc, FechaAlqu  
FROM Alquila  
WHERE FechaDev IS NULL  
ORDER BY FechaAlqu DESC
```

	IdSoc	FechaAlqu
1	5	2008-11-24 00:00:00
2	1	2008-11-24 00:00:00
3	2	2008-11-10 00:00:00
4	2	2008-11-10 00:00:00

**ix) Agrupar filas (GROUP BY, HAVING y funciones de agregado: Count(\*), Count (columna), Avg( ), Sum( ), Min( ), Max( )**

GROUP BY se usa para separar los registros seleccionados en grupos específicos, por ejemplo para mostrar cuántas películas hay de cada nacionalidad.

```
SELECT NacionalPe, Count (*) AS N° de ejemplares'
FROM Películas
GROUP BY NacionalPe
```

	NacionalPe	N° de ejemplares
1	Americana	1
2	Española	3
3	Japonesa	1

Muestra las películas hay de cada nacionalidad siempre que sean mayores o iguales que 3.

```
SELECT NacionalPe, Count (*) AS 'N° de ejemplares'
FROM Películas
GROUP BY NacionalPe
HAVING Count (*)>=3
```

	NacionalPe	N° de ejemplares
1	Española	3

Muestra el número de películas que tiene la tabla de películas.

```
SELECT Count (*) AS 'N° de películas'
FROM Películas
```

	N° de películas
1	5

Muestra cuantos títulos distintos hay en la tabla de películas.

```
SELECT Count (Titulo) AS 'Títulos distintos'
FROM Películas
```

	Títulos distintos
1	5

Mostrar la suma de los identificadores de las películas que tenemos:

```
SELECT Sum(IdPelícula) AS 'Suma IdPelícula'
FROM Películas
```

	Suma IdPelícula
1	15

Mostrar el número de películas que tenemos:

```
SELECT Count (*) AS 'N° Películas'
FROM Películas
```

	N° Películas
1	5

Mostrar la media de los identificadores de las películas que tenemos:

```
SELECT Avg (IdPelícula) AS 'Media IdPelícula'
FROM Películas
```

	Media IdPelícula
1	3

Mostrar el identificador más grande que tenemos:

```
SELECT Max (IdPelícula) AS 'IdPelícula Mayor'
FROM Películas
```

	IdPelícula Mayor
1	5

Mostrar el identificador más pequeño que tenemos:

```
SELECT Min (IdPelícula) AS 'IdPelícula Menor'
FROM Películas
```

	IdPelícula Menor
1	1

### c) Consultas basadas en más de una tabla

#### i) Producto cartesiano (FROM tabla1, tabla2)

Devuelve el producto Cartesiano de la tabla películas y directores.

```
SELECT *
FROM Películas, Directores
```

	IdPelícula	Título	NacionaliPe	Productora	FechaPe	CódDi	CódDi	NombreDi	NacionaliDi
1	1	La bala que dobló la es...	Española	Hermanos García S.A.	1989-10-23 ...	5	1	Pedro Almodovar	Español
2	2	Los otros	Española	Producciones El ag...	1956-12-12 ...	2	1	Pedro Almodovar	Español
3	3	Mujeres al borde de un...	Española	Maricomix S.L.	1963-12-05 ...	1	1	Pedro Almodovar	Español
4	4	CoCon	Americana	Columbia pictures	1963-05-08 ...	5	1	Pedro Almodovar	Español
5	5	Ni quito ni pongo	Japonesa	Yujara	1967-11-15 0...	5	1	Pedro Almodovar	Español
6	1	La bala que dobló la es...	Española	Hermanos García S.A.	1989-10-23 ...	5	2	Alejandro Amenavar	Español
7	2	Los otros	Española	Producciones El ag...	1956-12-12 ...	2	2	Alejandro Amenavar	Español

#### ii) Combinación común

##### (1) Con FROM t1, t2 WHERE t1.campox = t2.campoy

Devuelve los títulos de las películas con el Nombre de su director.

```
SELECT Título, NombreDi
FROM Películas, Directores
WHERE Películas.CódDi = Directores.CódDip
```

	Título	NombreDi
1	La bala que dobló la esquina	Mariano la Piedara
2	Los otros	Alejandro Amenavar
3	Mujeres al borde de un ataque de nervios	Pedro Almodovar
4	CoCon	Mariano la Piedara
5	Ni quito ni pongo	Mariano la Piedara

##### (2) Con INNER JOIN

Si la sentencia SELECT es la más utilizada para consulta de información, el operador JOIN es el más usado para obtener información de tablas relacionadas. Si deseamos obtener datos de dos tablas que están relacionadas por un atributo, deberemos utilizar este operador para obtener la información contenida en ambas.

Especifica que se devuelvan todos los pares de filas coincidentes. Descarta las filas no coincidentes de las dos tablas. Éste es el valor predeterminado si no se especifica ningún tipo de combinación.



Por ejemplo para obtener los títulos de las películas con el Nombre de su director.

```
SELECT Título, NombreDi
FROM Películas
INNER JOIN Directores ON Películas.CódDi = Directores.CódDi
```

	Título	NombreDi
1	La bala que dobló la esquina	Mariano la Piedara
2	Los otros	Alejandro Amenabar
3	Mujeres al borde de un ataque de nervios	Pedro Almodovar
4	CoCon	Mariano la Piedara
5	Ni quito ni pongo	Mariano la Piedara

Para mostrar el nombre de los actores, las películas y del director, de las películas dirigidas por 'Mariano la Piedara'.

```
SELECT NombreAc, Título, NombreDi
FROM Películas
INNER JOIN Directores
ON Películas.CódDi = Directores.CódDi
INNER JOIN Participa
ON Películas.IdPelícula = Participa.IdPelícula
INNER JOIN Actores
ON Participa.IdActor = Actores.IdActor
WHERE NombreDi = 'Mariano la Piedara'
```

	NombreAc	Título	NombreDi
1	Antonio Banderas	La bala que dobló la esquina	Mariano la Piedara
2	Lucía la Piedra	Ni quito ni pongo	Mariano la Piedara
3	Pilar Rubio	CoCon	Mariano la Piedara

Para mostrar el nombre de las películas, el nombre del socio, y la fecha de alquiler de las películas que están alquiladas.

```
SELECT Título, NombreSoc, FechaAlqu
FROM Películas
INNER JOIN Ejemplares
ON Películas.IdPelícula = Ejemplares.IdPelícula
INNER JOIN Alquiler
ON Ejemplares.IdEjemplar = Alquiler.IdEjemplar
INNER JOIN Socios
ON Alquiler.IdSoc = Socios.IdSoc
WHERE FechaDev IS NULL
```

	Título	NombreSoc	FechaAlqu
1	La bala que dobló la esquina	Francisco Mariano Prieto	2008-11-10 00:00:00
2	CoCon	Francisco Mariano Prieto	2008-11-10 00:00:00
3	La bala que dobló la esquina	Isaac Torralba	2008-11-24 00:00:00
4	Los otros	Francisco Barragán	2008-11-24 00:00:00

**(a) LEFT**

Especifica que todas las filas de la tabla de la izquierda que no cumplan la condición especificada se incluyan en el conjunto de resultados, además de todas las filas que devuelva la combinación interna. Las columnas de salida de la tabla de la izquierda se establecen a NULL.

Esta operación consiste en añadir al resultado del INNER JOIN las filas de la tabla de la izquierda que no tienen correspondencia en la otra tabla, y rellenar en esas filas los campos de la tabla de la derecha con valores nulos.

Por ejemplo para devolver todos nombres de actores y las películas que protagonizó aunque no tengan ninguna película, a estas se les pone el valor NULL.

```
SELECT NombreAc, Título
FROM Actores
LEFT JOIN Participa ON
    Participa.IdActor = Actores.IdActor
LEFT JOIN Películas
ON Películas.IdPelícula = Participa.IdPelícula
```

	NombreAc	Título
1	Antonio Banderas	La bala que dobló la esquina
2	Antonio Banderas	Los otros
3	Antonio Banderas	Mujeres al borde de un ataque de nervios
4	Patricia Conde	Los otros
5	Lucía la Piedra	Ni quito ni pongo
6	Pilar Rubio	CoCon
7	Brad Pitt	NULL
8	Robert de Niro	NULL
9	Jorge Javier Vazquez	NULL

**(b) RIGHT**

Especifica que todas las filas de la tabla de la derecha que no cumplan la condición especificada se incluyan en el conjunto de resultados, además de las que devuelva la combinación interna. Las columnas de salida de la tabla de la derecha se establecen a NULL.

Esta operación consiste en añadir al resultado del INNER JOIN las filas de la tabla de la derecha que no tienen correspondencia en la otra tabla, y rellenar en esas filas los campos de la tabla de la izquierda con valores nulos.

Por ejemplo para devolver todos los nombres de directores y las películas que dirigió aunque no tenga ninguna película, a estas se les pone el valor NULL.

```
SELECT Título, NombreDi
FROM Películas
RIGHT JOIN Directores
ON Directores.CódDi = Películas.CódDi
```

	Título	NombreDi
1	Mujeres al borde de un ataque de nervios	Pedro Almodovar
2	Los otros	Alejandro Amenabar
3	NULL	Antonio Banderas
4	NULL	Hermanos Cohen
5	La bala que dobló la esquina	Mariano la Piedara
6	CoCon	Mariano la Piedara
7	Ni quito ni pongo	Mariano la Piedara

### (c) FULL

Si una fila de la tabla de la izquierda o de la derecha no coincide con los criterios de selección, especifica que la fila se incluya en el conjunto de resultados y las columnas de resultados que corresponden a la otra tabla se establezcan como NULL. Se trata de una adición a todas las filas que normalmente devuelve la combinación interna.

Por ejemplo para devolver todas las filas de las dos tabla, tanto las comunes como las no comunes.

```
SELECT Alquila.FechaDev, Título
FROM Alquila
FULL JOIN Películas
ON Películas.IdPelícula = Alquila.IdPelícula
```

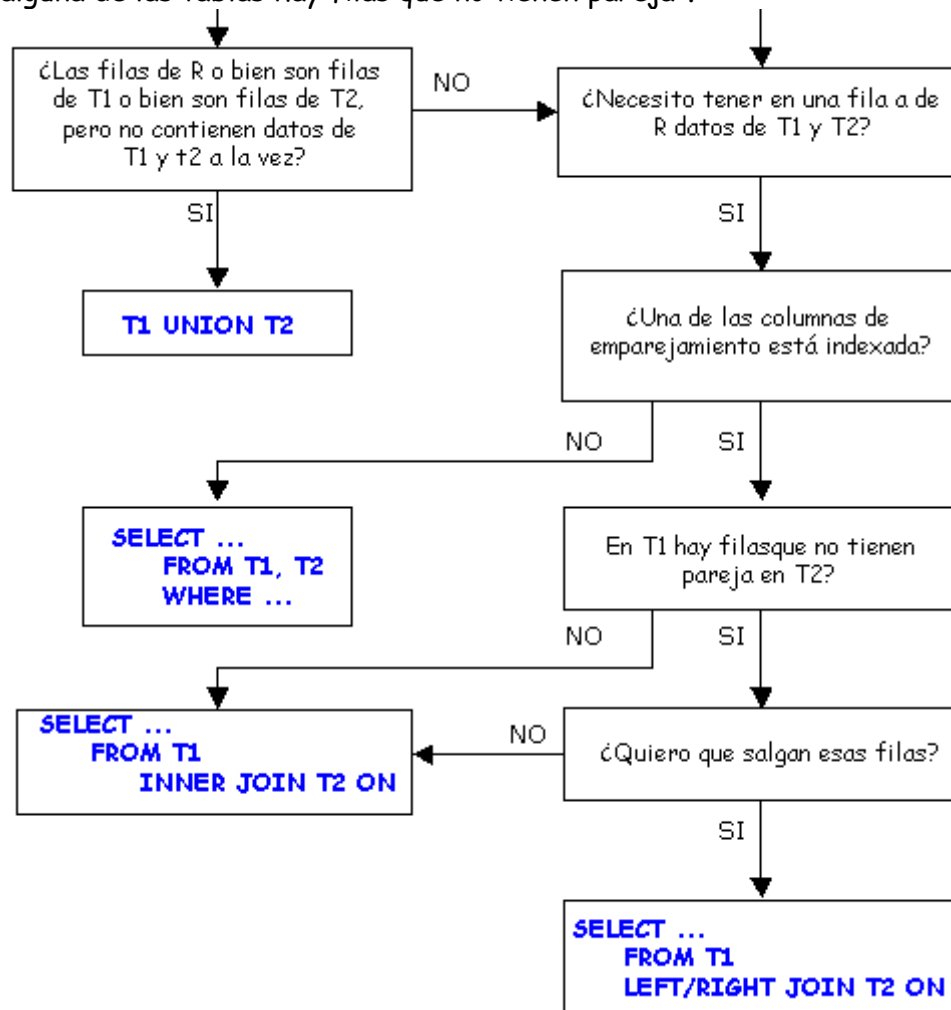
	FechaDev	Título
1	NULL	La bala que dobló la esquina
2	2008-11-15 00:00:00	La bala que dobló la esquina
3	2008-11-15 00:00:00	La bala que dobló la esquina
4	2008-11-20 00:00:00	La bala que dobló la esquina
5	2008-11-27 00:00:00	La bala que dobló la esquina
6	NULL	La bala que dobló la esquina
7	2008-11-26 00:00:00	La bala que dobló la esquina
8	2008-11-26 00:00:00	La bala que dobló la esquina
9	2008-11-27 00:00:00	La bala que dobló la esquina
10	NULL	Los otros
11	2008-11-22 00:00:00	Los otros
12	2008-11-26 00:00:00	Mujeres al borde de un ataque de nervios
13	2008-11-27 00:00:00	CoCon
14	NULL	CoCon
15	2008-11-26 00:00:00	Ni quito ni pongo

### (d) Resumen de cuándo utilizar cada operación.

Para saber en cada caso qué tipo de operación se debe utilizar, a continuación tienes un gráfico que indica qué preguntas se tienen que hacer y según la respuesta, qué operación utilizar.

Para resumir hemos llamado T1 y T2 las tablas de las que queremos sacar los datos y R la tabla que representa el resultado de consulta. T1 y T2 podrían ser tablas temporales o consultas.

En la última parte cuando se pregunta "En T1 hay filas que no tienen pareja en T2", la pregunta se debe de interpretar como "en alguna de las tablas hay filas que no tienen pareja".



### iii) Combinación con alguna condición más (AND condición)

Para devolver el nombre del actor y las películas en las que participa y que además el actor sea 'Antonio Banderas'.

```

SELECT NombreAc, Título
FROM Actores
  INNER JOIN Participa
    ON Participa.IdActor = Actores.IdActor
  INNER JOIN Películas
    ON Películas.IdPelícula = Participa.IdPelícula AND
    Actores.NombreAc = 'Antonio Banderas'
  
```

	NombreAc	Título
1	Antonio Banderas	La bala que dobló la esquina
2	Antonio Banderas	Los otros
3	Antonio Banderas	Mujeres al borde de un ataque de nervios

#### iv) Unión de filas (UNION)

Mezcla los resultados de dos o más consultas en un solo conjunto de resultados que contiene todas las filas que pertenecen a las consultas de la unión. Este procedimiento es distinto de la utilización de combinaciones de columnas de dos tablas.

Dos reglas básicas para combinar los conjuntos de resultados de dos consultas con UNION son:

El número y el orden de las columnas deben ser idénticos en todas las consultas.

Los tipos de datos deben ser compatibles.

Por ejemplo para mostrar las nacionalidades de las que hay algún actor o algún director. (Unión)

```
SELECT NacionaliaC
FROM Actores
UNION
SELECT NacionaliaDi
FROM Directores
```

	NacionaliaC
1	Americano
2	Español
3	Española
4	Polaco

Por ejemplo para mostrar una tabla con los nombres de los Actores y de los directores, además le añadimos el tipo según sea un actor o director.

```
SELECT 'Tipo' = 'Actor', NombreAc
FROM Actores
UNION
SELECT 'Tipo' = 'Director', NombreDi
FROM Directores
```

	Tipo	NombreAc
1	Actor	Antonio Banderas
2	Actor	Brad Pitt
3	Actor	Jorge Javier Vazquez
4	Actor	Lucía la Piedra
5	Actor	Patricia Conde
6	Actor	Pilar Rubio
7	Actor	Robert de Niro
8	Director	Alejandro Amenabar
9	Director	Antonio Banderas
10	Director	Hermanos Cohen
11	Director	Mariano la Piedra
12	Director	Pedro Almodovar

**v) Diferencia (NOT EXISTS, NOT IN)**

NOT EXISTS funciona igual que EXISTS, salvo por el hecho de que la cláusula WHERE en la que se utiliza se cumple si la subconsulta no devuelve ninguna fila.

Devuelve los empleados que no hayan echo ningún pedido.

```
SELECT empleado.dni
FROM empleado
WHERE NOT EXISTS (
    SELECT pedidos.dni_emp
    FROM pedidos
    WHERE empleado.dni = pedidos.dni_emp
)
```

Por ejemplo para mostrar los ejemplares que no han sido alquilados. (Diferencia)

```
SELECT IdEjemplar
FROM Ejemplares
WHERE IdEjemplar NOT IN (
    SELECT DISTINCT IdEjemplar
    FROM Alquila
)
```

	IdEjemplar
1	6
2	8
3	9
4	10
5	11
6	12
7	15
8	16
9	18
10	19
11	21
12	22

Por ejemplo para mostrar las fechas en que se ha alquilado algún ejemplar y no se ha devuelto ninguno. (Diferencia)

```
SELECT FechaAlqu
FROM Alquila
WHERE FechaAlqu NOT IN (
    SELECT FechaAlqu
    FROM Alquila
    WHERE FechaDev IS NULL
)
```

	FechaAlqu
1	2008-11-12 00:00:00
2	2008-11-12 00:00:00
3	2008-11-20 00:00:00
4	2008-11-21 00:00:00
5	2008-11-23 00:00:00
6	2008-11-23 00:00:00
7	2008-11-27 00:00:00

Por ejemplo para mostrar las nacionalidades de las que hay algún actor y no hay algún director. (Diferencia):

```
SELECT DISTINCT NacionalAc
FROM Actores
WHERE NacionalAc NOT IN (
    SELECT NacionalDi
    FROM Directores
)
```

NacionalAc
1 Española

Por ejemplo para mostrar las Nacionalidades de los directores que no estén en actores. (Diferencia)

```
SELECT DISTINCT NacionalDi
FROM Directores
WHERE NacionalDi NOT IN (
    SELECT DISTINCT NacionalAc
    FROM Actores
)
```

NacionalDi
1 Polaco

Por ejemplo para mostrar los códigos de los directores que no han dirigido ninguna película. (Diferencia)

```
SELECT CódDi
FROM Directores
WHERE NOT EXISTS (
    SELECT CódDi
    FROM Películas
    WHERE Directores.CódDi = CódDi
)
```

O bien:

```
SELECT CódDi
FROM Directores
WHERE CódDi <> ALL (
    SELECT DISTINCT CódDi
    FROM Películas
)
```

CódDi
1 3
2 4

**vi) Intersección (EXISTS, INNER JOIN)**

Especifica una subconsulta para probar la existencia de filas.

Por ejemplo la siguiente consulta devuelve los socios que han realizado algún alquiler. (Intersección)

```
SELECT NombreSoc
FROM Socios
WHERE EXISTS (
    SELECT *
    FROM Alquiler
    WHERE Socios.IdSoc = Alquiler.IdSoc
)
```

	NombreSoc
1	Isaac Torralba
2	Francisco Mariano Prieto
3	Felipe Reyes
4	David Rodríguez
5	Francisco Barragán
6	Antonio Megías
7	Azucena Benítez
8	ángela López

Por ejemplo para mostrar los Ejemplares que han sido alquilados. (Intersección)

```
SELECT IdEjemplar
FROM Ejemplares
WHERE IdEjemplar IN (
    SELECT DISTINCT IdEjemplar
    FROM Alquiler
)
```

	IdEjemplar
1	1
2	2
3	3
4	4
5	5
6	7
7	13
8	14
9	17
10	20

Por ejemplo para mostrar las fechas en que han alquilado algún ejemplar y se ha devuelto alguno también. (Intersección)

```
SELECT FechaAlqu
FROM Alquiler
WHERE FechaAlqu IN (
    SELECT FechaDev
    FROM Alquiler
)
```

	FechaAlqu
1	2008-11-20 00:00:00
2	2008-11-27 00:00:00



Por ejemplo para mostrar las nacionalidades de las que hay algún actor y hay algún director. (Intersección)

```
SELECT DISTINCT Nacionalidad
FROM Actores
WHERE Nacionalidad IN (
    SELECT Nacionalidad
    FROM Directores
)
```

Nacionalidad	
1	Americano
2	Español

Por ejemplo para mostrar el código de los directores que han dirigido alguna película. (Intersección)

```
SELECT CódDi
FROM Directores
WHERE CódDi = ANY (
    SELECT DISTINCT CódDi
    FROM Películas
)
```

CódDi	
1	1
2	2
3	5

### vii) División

Por ejemplo para saber el actor o los actores que han participado en las tres películas especificadas.

```
-- Para hacer el divisor
CREATE VIEW Películas2 AS (
    SELECT DISTINCT Películas.IdPelícula
    FROM Películas, Participa
    WHERE Películas.IdPelícula = Participa.IdPelícula AND
        Título IN ('La bala que dobló la esquina', 'Los otros',
            'Mujeres al borde de un ataque de nervios')
)
GO
```

Nos crearía una vista que contiene los siguientes resultados:

IdPelícula	
1	1
2	2
3	3

Y ahora para que nos devolviera que actores están en las tres películas efectuaríamos la división mediante las siguientes instrucciones:

```
-- Saca el resultado
SELECT DISTINCT IdActor, NombreAc
  FROM Actores
 WHERE NOT EXISTS (
    SELECT *
      FROM Películas2
     WHERE NOT EXISTS (
        SELECT *
          FROM Participa
         WHERE Participa.IdActor = Actores.IdActor And
              Participa.IdPelícula = Películas2.IdPelícula
      )
  )
```

	IdActor	NombreAc
1	1	Antonio Banderas

#### d) Consulta de creación de una nueva tabla (SELECT INTO tabla)

Crea una tabla con todos los empleados cuyo sueldo sea mayor a 3000.

```
SELECT dni, nombre, sueldo
      INTO Emp_bien_pagado
  FROM empleado
 WHERE sueldo > 3000
```

#### e) Consulta en la creación de una vista (CREATE VIEW AS)

Para crear una vista debemos utilizar la sentencia CREATE VIEW, debiendo proporcionar un nombre a la vista y una sentencia SQL SELECT válida.

```
CREATE VIEW NombreVista AS (SentenciaSELECT)
```

Este ejemplo crea una vista sobre la tabla películas, en la que se nos muestren los distintos títulos que tenemos en nuestro VideoClub, se haría de la siguiente forma.

```
CREATE VIEW DisTítulos AS (
  SELECT DISTINCT Título
  FROM Películas
)
```

Vistas
dbo.DisTítulos

La vista se puede usar, por ejemplo para mostrar los títulos de las películas anteriores, podríamos escribir:

```
SELECT * FROM DisTítulos
```

	Título
1	La bala que dobló la esquina
2	Los otros
3	Mujeres al borde de un ataque de nervios
4	CoCon
5	Ni quito ni pongo

Por ejemplo para crear una vista llamada 'películas\_prestadas' donde se muestre nombre de las películas, el nombre del socio, y la fecha de alquiler de las películas que están alquiladas:

```
CREATE VIEW películas_prestadas AS (
    SELECT Título, NombreSoc, FechaAlqu
    FROM Películas
    INNER JOIN Ejemplares
    ON Películas.IdPelícula = Ejemplares.IdPelícula
    INNER JOIN Alquila
    ON Ejemplares.IdEjemplar = Alquila.IdEjemplar
    INNER JOIN Socios
    ON Alquila.IdSoc = Socios.IdSoc
    WHERE FechaDev IS NULL
)
```

Vistas	labo.películas_prestadas
--------	--------------------------

La vista se puede usar, por ejemplo para mostrar los títulos de las películas anteriores, podríamos escribir:

```
SELECT * FROM películas_prestadas
```

	Título	NombreSoc	FechaAlqu
1	La bala que dobló la esquina	Francisco Mariano Prieto	2008-11-10 00:00:00
2	CoCon	Francisco Mariano Prieto	2008-11-10 00:00:00
3	La bala que dobló la esquina	Isaac Torralba	2008-11-24 00:00:00
4	Los otros	Francisco Barragán	2008-11-24 00:00:00

## f) Subconsultas

Una subconsulta es una sentencia SELECT que aparece dentro de otra sentencia SELECT. Normalmente se utilizan para filtrar una cláusula WHERE o HAVING con el conjunto de resultados de la subconsulta, aunque también pueden utilizarse en la lista de selección.

Por ejemplo podríamos consultar el nombre del socio que realizó el último alquiler. En este caso, la subconsulta se ejecuta en primer lugar, obteniendo el valor de la máxima fecha de alquiler, y posteriormente se obtienen los datos de la consulta principal.

```
SELECT NombreSoc
FROM Soios
WHERE IdSoc = (
    SELECT Max (FehaAlqu)
    FROM Alquila
)
```

	NombreSoc
1	Isaac Torralba

Una subconsulta tiene la misma sintaxis que una sentencia SELECT normal exceptuando que aparece encerrada entre paréntesis.

La subconsulta se puede encontrar en la lista de selección, en la cláusula WHERE o en la cláusula HAVING de la consulta principal.

Tiene las siguientes restricciones:

- No puede contener la cláusula ORDER BY
- No puede ser la UNION de varias sentencias SELECT
- Si la subconsulta aparece en la lista de selección, o esta asociada a un operador igual "=" solo puede devolver un único registro.

### i) Referencias externas

A menudo, es necesario, dentro del cuerpo de una subconsulta, hacer referencia al valor de una columna de la fila actual en la consulta principal, ese nombre de columna se denomina referencia externa.

Una referencia externa es un campo que aparece en la subconsulta pero se refiere a la una de las tablas designadas en la consulta principal.

Cuando se ejecuta una consulta que contiene una subconsulta con referencias externas, la subconsulta se ejecuta por cada fila de la consulta principal.

En este ejemplo la subconsulta aparece en la lista de selección, ejecutándose una vez por cada fila que devuelve la consulta principal, así para mostrar el nombre, la fecha del último alquiler y la dirección de todos los socios.

```
SELECT TOP 1 NombreSoc, (
    SELECT Max (FechaAlqu)
    FROM Alquila
    WHERE IdSoc = Socios.IdSoc) AS 'Fecha último alquiler',
    DireccSoc
FROM Socios
```

	NombreSoc	Fecha último alquiler	DireccSoc
1	Isaac Torralba	2008-11-27 00:00:00	C/ Real, 3 - Santa Fe

## ii) Anidar subconsultas

Las subconsultas pueden anidarse de forma que una subconsulta aparezca en la cláusula WHERE (por ejemplo) de otra subconsulta que a su vez forma parte de otra consulta principal.

Por ejemplo para mostrar el título y la productora de las películas alquiladas y devueltas.

```
SELECT Título, Productora
FROM Películas
WHERE IdPelícula IN (
    SELECT IdPelícula
    FROM Ejemplares
    WHERE IdEjemplar IN (
        SELECT IdEjemplar
        FROM Alquila
        WHERE FechaDev IS NULL
    )
)
```

	Título	Productora
1	La bala que dobló la esquina	Hermanos García S.A.
2	Los otros	Producciones El aguila real
3	CoCon	Columbia pictures

Por ejemplo para mostrar el título y la productora de los ejemplares, haríamos:

```
SELECT Título, Productora
FROM Películas
WHERE IdPelícula IN (
    SELECT IdPelícula
    FROM Ejemplares
    WHERE Películas.IdPelícula = Ejemplares.IdPelícula
)
```

Pero podría haberse escrito como:

```
SELECT Título, Productora
FROM Películas, Ejemplares
WHERE Películas.IdPelícula = Ejemplares.IdPelícula
```

Los resultados que se obtienen con subconsultas normalmente pueden conseguirse a través de consultas combinadas (JOIN). De hecho la consulta anterior también se podría haber escrito como:

```
SELECT Título
FROM Películas
INNER JOIN Ejemplares
ON Películas.IdPelícula = Ejemplares.IdPelícula
```

	Título	Productora
1	La bala que dobló la esquina	Hermanos García S.A.
2	Los otros	Producciones El aguila real
3	Mujeres al borde de un ataque de nervios	Maricomix S.L.
4	CoCon	Columbia pictures
5	Ni quito ni pongo	Yujara

Normalmente es más rápido utilizar un JOIN en lugar de una subconsulta, aunque esto depende sobre todo del diseño de la base de datos y del volumen de datos que tenga.

## 8) Variables

La declaración de variables en SQL Server utilizando T-SQL es simple, se utiliza la instrucción `DECLARE` para indicar que se está declarando una variable, después se pondrá el nombre de la variable con una arroba como prefijo, después del nombre irá el tipo de dato, este tipo de dato es el mismo que se utiliza en la definición de los campos de una tabla de SQL Server.

El siguiente es un ejemplo de declaración de dos variables, una del tipo `varchar` de 20 caracteres de longitud, y la otra una declaración de una variable del tipo `int`:

```
DECLARE @Nombre varchar (20)
DECLARE @NúmPelículas int
```

El mecanismo para asignar un valor inicial a la variable se hace de dos maneras, una es con la instrucción `SET` si es que se quiere asignar el valor directamente y la otra es con la instrucción `SELECT` si se quiere asignar la variable con un valor consultado de alguna tabla. Veamos algunos ejemplos de esto.

En este caso un ejemplo de asignación con `SET`, para asignar un valor directamente:

```
DECLARE @Nombre1 varchar (20)
SET @Nombre1 = 'Isaac Torralba'
PRINT @Nombre1           -- Muestra el valor de la variable Nombre1 como texto.
SELECT @Nombre1          -- Muestra una tabla con el valor de la variable Nombre1.
```

Ahora un ejemplo con variables numéricas:

```
DECLARE @a int
SET @a=7
SELECT @a
```

Otro ejemplo sería este mediante un `SELECT`, para asignar un valor consultado de la tabla `empleados`:

```
DECLARE @Nombre2 varchar (50)
SELECT @Nombre2 = NombreEm FROM Empleados WHERE CódigoEm = 2
PRINT @Nombre2           -- Muestra el valor de la variable Nombre2 como texto.
```

O este otro ejemplo mediante `SELECT`, para asignar el número de socios que disponemos a una variable:

```
DECLARE @NúmEmpleados int
SELECT @NúmEmpleados = Count (*) FROM Empleados
PRINT 'Hay ' + Cast(@NúmEmpleados AS varchar) + ' empleados.'
-- o bien
PRINT 'Hay ' + CONVERT (varchar, @NúmEmpleados) + ' Empleados.'
-- Ambas muestran el texto 'Hay 8 socios.'
```

Otro ejemplo de conversiones mediante cast podría ser el siguiente:

```
DECLARE @b char (1)
SELECT @b='5'
PRINT @b + ' -->' + Cast(@a AS varchar (1))
PRINT Cast(@b AS int) + @a
```

Las funciones CAST y CONVERT convierten un valor (una variable local, una columna u otra expresión) de un tipo de datos a otro.

De esta manera es como se asignan valores a una variable, ahora bien, si queremos utilizar esta variable para pasar valores a una consulta o instrucción, lo haríamos sustituyendo el valor fijo por el nombre de la variable, así, podríamos variar los valores en la variable y mantener la consulta o la instrucción SQL sin cambios.

Veamos un ejemplo de uso:

```
DECLARE @Nombre3 varchar (20)
SET @Nombre3 = 'Isaac Torralba'

SELECT *
FROM Socios
WHERE NombreSoc = @Nombre3
```

	IdSoc	NombreSoc	DireccSoc	TeléfonoSoc	SocioAvala
1	1	Isaac Torralba	C/ Real, 3 - Santa Fe	958741236	1

Nos preguntamos qué razón tendría utilizar variables si se pueden utilizar los valores directamente en la consulta, para este caso, pues ninguna ya que en este caso si se trata de consultas de prueba en la consola de consultas de SQL Server pues es mejor utilizar los valores directamente, sin embargo, cuando hablamos de procedimientos almacenados o de instrucciones incluidas en un CURSOR o en un ciclo donde los valores cambian a cada vuelta o cambian por el paso de valores, pues ahí sí tendría sentido.

Existen también otro tipo de variables en T-SQL, que son las variables globales. Éstas están definidas en SQL Server, no es posible definir variables globales dentro de nuestras rutinas, solamente pueden utilizarse las variables ya definidas. Estas variables están siempre predefinidas con los símbolos @@ precediendo al nombre. Nunca se deben declarar variables locales con el mismo nombre que una global porque puede tener resultados inesperados.

Algunos ejemplos de estas variables globales son:

Variable	Acción
@@CONNECTIONS	Conexiones totales intentadas.
@@DBTS	Valor del sello temporal único para la base de datos.
@@ERROR	Ultimo número de error del sistema.
@@IDENTITY	El último valor de entidad insertado.
@@IO_BUSY	Tiempo acumulado de Entrada/Salida del servidor.
@@LANGID	ID del lenguaje actual.
@@LANGUAGE	Nombre del lenguaje actual.
@@MAX_CONNECTIONS	Máximo número de conexiones.
@@MAX_PRECISION	Nivel de precisión para tipos de datos decimales numéricos.
@@MICROSOFTVERSION	Número de versión interno de SQL Server.



Variable	Acción
@@NESTLEVEL	Nivel de anidamiento de subrutinas, entre 1 y 16.
@@PROCID	Identidad del proceso almacenado en curso.
@@ROWCONT	Número de filas afectadas por la última consulta.
@@SERVERNAME	Nombre del servidor local.
@@SERVICENAME	Nombre del servicio que se está ejecutado.
@@SPID	Identidad del proceso del servidor en curso.
@@TEXTSIZE	Máximo en curso para los datos de tipo text o imagen, con un valor por omisión de 4K.
@@TIMETICKS	Número de microsegundos por ticks, independiente de la máquina, un ticks es igual a 31.25 milisegundos o 1/32 segundos.
@@TOTAL_READ	Número de lecturas de disco (sin reserva).
@@TOTAL_WRITE	Número de escrituras en disco.
@@TRANCOUNT	Transacciones actuales de usuarios activos.
@@SERVERNAME	Nombre del servidor.
@@SERVICENAME	Nombre del servicio.
@@VERSION	Fecha y Versión de SQL Server.

Para su utilización podríamos realizar alguna acción parecida a la siguiente:

```
SELECT  @@VERSION          AS 'Versión de SQL Server',
        @@SERVERNAME      AS 'Nombre del servidor',
        @@SERVICENAME     AS 'Nombre del servicio'
```

	Versión de SQL Server	Nombre del servidor	Nombre del servicio
1	Microsoft SQL Server 2000 - 8.00.194 (Intel X...	PC1	MSSQLSERVER

```
PRINT @@SERVERNAME
```

Que escribiría el texto PC1.

### a) Cast y Convert

Convierten una expresión de un tipo de datos en otro de forma explícita.

CAST y CONVERT proporcionan funciones similares.

```
CONVERT ( TipoDato [ ( Longitud) ] , Expresión [ , Estilo ] )
```

Donde:

- TipoDato, es el tipo de destino al que queremos convertir la expresión
- Expresión, la expresión que queremos convertir
- Estilo, parámetro opcional que especifica el formato que tiene expresión. Por ejemplo, si queremos convertir un varchar a datetime, aquí debemos especificar el formato de la fecha (el tipo varchar).

Por ejemplo vamos a convertir un valor varchar a datetime. El 103 indica el formato en el que está escrita la fecha es dd/mm/aa.

```
DECLARE @fecha varchar (20)
SET @fecha = Convert (datetime, '19/03/2008',103)
PRINT @fecha
```

Devolviendo Dic 11 2008 12:00AM

Convertimos ahora una fecha a varchar y la formateamos. El 3 indica el formato (dd/mm/aa).

```
DECLARE @fecha datetime, @fechaFormateada varchar (20)
SET @fecha = GetDate ()
SET @fechaFormateada = Convert (varchar (20), @fecha, 3)
PRINT @fechaFormateada
```

Devolviendo la fecha actual (11/12/08) como un texto.

Un ejemplo utilizando CAST, para convertir un varchar a int:

```
/* Un ejemplo utilizando CAST, para convertir un varchar a int:*/
DECLARE @dato1 varchar (2), @dato2 int
SET @dato1 = '27'
SET @dato2 = Cast(@dato1 AS int)
SELECT @dato2 AS Valor
```



A continuación mostramos la tabla de códigos de estilo (obtenida de Microsoft).

Sin el siglo (yy)	Con el siglo (yyyy)	Estándar	Entrada/Salida**
-	0 o 100 (*)	Valor predeterminado	mon dd yyyy hh:miAM (o PM)
1	101	USA	mm/dd/yy
2	102	ANSI	yy.mm.dd
3	103	Británico/Francés	dd/mm/yy
5	105	Italiano	dd-mm-yy
6	106	-	dd mes aa
7	107	-	Mes dd, aa
8	108	-	hh:mm:ss
-	9 o 109 (*)	Predeterminado + milisegundos	mon dd yyyy hh:mi:ss:mmmAM (o PM)
10	110	USA	mm-dd-yy
11	111	JAPÓN	yy/mm/dd
12	112	ISO	yymmdd
-	13 o 113 (*)	Europeo predeterminado + milisegundos	dd mon yyyy hh:mm:ss:mmm(24h)
14	114	-	hh:mi:ss:mmm(24h)

\* Los valores predeterminados (estilo 0 a 100, 9 a 109, 13 ó 113) siempre devuelven el siglo (yyyy).

Para borrar los tipos creados usamos:

```
EXEC sp_droptype 'ejemplo1'
EXEC sp_droptype 'nss'
EXEC sp_droptype 'cumpleaños'
EXEC sp_droptype 'telefono'
EXEC sp_droptype 'fax'
```

## 9) Funciones

Microsoft agregó nuevas características a su producto SQL 2005, y lo más interesante para los programadores del SQL es la posibilidad de hacer funciones definidas por el usuario. La adición de funciones al lenguaje del SQL solucionara los problemas de reutilización del código y sino mayor flexibilidad al programar las consultas de SQL.

### a) Funciones del sistema

Las funciones del sistema se pueden utilizar para obtener información acerca de nuestro sistema de computadora, acerca de los usuarios, y objetos de la base de datos en general. Las funciones del sistema se pueden usar en las cláusulas SELECT y WHERE.

Función	Acción
HOST_NAME()	Nombre de la computadora servidor.
HOST_ID()	Nro. de ID de la computadora servidor.
SUSER_ID( <i>número_de_conexion</i> )	Nro. de conexión de usuario.
SUSER_NAME( <i>id_de_usuario_servidor</i> )	Nombre de conexión de usuario.
USER_ID( <i>nombre_de_usuario</i> )	Nro. ID del usuario para la base de datos.
USER_NAME( <i>id_usuario</i> )	Nombre de usuario para la base de datos.
DB_NAME( <i>id_datos</i> )	Nombre de la base de datos.
DB_ID( <i>nombre_bdd</i> )	Nro. de ID de la base de datos.
GETANSINULL( <i>nombre_dbb</i> )	Vale 1 si se admite NULL de ANSI.
OBJECT_ID( <i>nombre de objeto</i> )	Nro. de objeto de la base de datos.
OBJECT_NAME( <i>id_objeto</i> )	Nombre de un objeto de la base de datos.
INDEX_COL( <i>nombre_tabla,id_indice,id_clave</i> )	Nombre de la columna índice.
COL_LENGTH( <i>nombre_tabla,nombre_columna</i> )	Longitud definida de una columna.
COL_NAME( <i>id_tabla,id_columna</i> )	Nombre de la columna.
DATALENGTH( <i>expresión</i> )	Longitud real de una expresión de un tipo de datos.
STATS_DATE( <i>id_tabla,id_indice</i> )	Fecha en que se actualizaron por última vez las estadísticas del índice ( <i>index_id</i> ).
COALESCE( <i>expresión1,expresión2,expresiónN</i> )	Proporciona la primera expresión no nula.
ISNULL( <i>expresión,valor</i> )	Cambia las entradas NULL por valor.
NULLIF( <i>expresión1,expresión2</i> )	Proporciona NULL cuando <i>expresión1</i> es NULL y cuando <i>expresión1</i> es igual a <i>expresión2</i> .

Algunos ejemplos de uso de funciones de fecha podrían ser los siguientes:  
Para mostrar el día de la fecha.

```
PRINT Day ( GetDate() )           -- Devolviendo en este caso 11
```

Para mostrar el día de una cadena de caracteres.

```
PRINT Day ('20/12/2008')         -- Devolviendo en este caso 20
```

Para mostrar el día de la semana de hoy.

```
PRINT DateName (dw, GetDate() )  -- Devolviendo en este caso 'Jueves'
```

Un ejemplo de uso de funciones de cadena podrían ser los siguientes:

Para mostrar un carácter ASCII.

```
PRINT Ascii ('a')           -- Devolviendo en este caso 97
PRINT Char (97)             -- Devolviendo en este caso 'a'
```

Para mostrar la longitud de una cadena.

```
PRINT Len ('Buenos días')   -- Devolviendo en este caso 11
```

Para reemplazar las 'o' de 'Hola' por 'e'.

```
PRINT Replace ('Hola', 'o', 'e') -- Devolviendo en este caso 'Hela'
```

Para mostrar una subcadena.

```
PRINT SubString ('Hola', 1, 3) -- Devolviendo en este caso 'ola'
```

Un ejemplo de uso de funciones matemáticas podrían ser los siguientes:

Para mostrar números aleatorios.

```
PRINT Rand ()               -- Devolviendo en este momento 0.894419
                             -- y en otro por ejemplo 0.0885662
```

Un ejemplo de la función IF, podría ser el que cuente el número de empleados que se llaman 'Nombre del empleado 1' y si existe alguno me dice si está o no.

```
IF (SELECT Count (*)
     FROM Empleados
     WHERE NombreEm='Nombre del empleado 1')>0
BEGIN
    PRINT 'Sí Está'
END
ELSE
BEGIN
    PRINT 'No está'
END
```

O bien

```
IF EXISTS (SELECT Count (*)
           FROM Empleados
           WHERE NombreEm = 'Nombre del empleado 1')
BEGIN
    PRINT 'Sí Está'
END
ELSE
BEGIN
    PRINT 'No está'
END
```

Indicándonos en este caso que sí está.

## b) Funciones definidas por el usuario

El servidor 2005 del SQL utiliza tres tipos de funciones: las funciones escalares, tabla en línea, funciones de tabla de multisentencias. Los tres tipos de funciones aceptan parámetros de cualquier tipo excepto el rowversion. Las funciones escalares devuelven un solo valor, tabla en línea y multisentencias devuelven un tipo de dato tabla.

### i) Funciones Escalares

Las funciones escalares vuelven un tipo de los datos tal como int, money, varchar, real, etc. Pueden ser utilizadas en cualquier lugar incluso incorporado dentro de sentencias SQL. La sintaxis para una función escalar es la siguiente:

```
CREATE FUNCTION <NombreFunción, sysname, FunctionName>
(
    -- Lista de parámetros
    <@Param1, sysname, @p1> <Data_Type_For_Param1, , int>, ...
)
-- Tipo de datos que devuelve la función.
RETURNS <Function_Data_Type, ,int>
AS
BEGIN
    ...
END
```

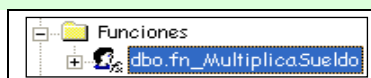
El siguiente ejemplo muestra como crear una función escalar.

```
CREATE FUNCTION fn_MultiplicaSueldo
(
    @Id                smallint,
    @Multiplicador      decimal
)
RETURNS decimal
AS
BEGIN
    DECLARE @Valor      decimal,
            @Salida      decimal

    SELECT @Valor = SueldoEm
    FROM Empleados
    WHERE CódigoEm = @Id

    SET @Salida = @Valor * @Multiplicador

    RETURN @Salida
END
```



Pueden ser utilizadas en cualquier sentencia Transact SQL. Un aspecto a tener en cuenta, es que para utilizar una función escalar debemos identificar el nombre de la función con el propietario de la misma.

El siguiente ejemplo muestra como utilizar la función anteriormente creada en una sentencia Transact SQL. Un aspecto muy a tener en cuenta es que la función ejecutará sus sentencias SELECT una vez por cada fila del conjunto de resultados devuelto por la consulta SELECT principal.

```
SELECT NombreEm, SueldoEm,
       dbo.fn_MultiplicaSueldo (CódigoEm, 10.10) AS Resultado
FROM Empleados
```

	NombreEm	SueldoEm	Resultado
1	Nombre del empleado 1	100.0000	1000.00
2	Nombre del empleado 2	200.0000	2000.00
3	Nombre del empleado 3	300.0000	3000.00
4	Nombre del empleado 4	400.0000	4000.00
5	Nombre del empleado 5	500.0000	5000.00

El siguiente ejemplo muestra como utilizar una función escalar en un script Transact SQL.

```
DECLARE @Id smallint, @Resultado1 decimal
SET @Id = 1
SET @Resultado = dbo.fn_MultiplicaSueldo (@Id, 10.0)
SELECT @Resultado AS Valor
```

Valor
1000

Por ejemplo para crear una función a la que pasamos un título de una película y devuelve el valor del nombre del director de la película\*/

```
CREATE FUNCTION DirectorPelícula ( @Título varchar(50) )
RETURNS varchar (50)
AS
BEGIN
    DECLARE @Director varchar (50)
    SELECT @Director = NombreDi
    FROM Películas
    INNER JOIN Directores
    ON Películas.CódDi = Directores.CódDi
    WHERE Título = @Título
    RETURN @Director
END
```

Para comprobar el funcionamiento de esta función haríamos:

```
PRINT dbo.DirectorPelícula ('Cocon')
```

Devolviendo 'Mariano la Piedara'. Para borrar la función creada haríamos:

```
DROP FUNCTION AñosPelícula
```

Por ejemplo para crear una función a la que pasamos un título de una película y devuelve el número de años en que se rodó la película

```
CREATE FUNCTION AñosPelícula ( @Título varchar(50),  
                                @FechaActual smalldatetime )  
  
    RETURNS    varchar (50)  
AS  
BEGIN  
    DECLARE @Años smallint, @Hoy smallint  
    --SELECT @Años = Year(@FechaActual) - Year (FechaPe)  
    -- O bien:  
    SELECT @Años = DATEDIFF (year, FechaPe, @FechaActual)  
        FROM Películas  
        WHERE Título = @Título  
    RETURN @Años  
END
```

Para comprobar el funcionamiento de esta función haríamos:

```
PRINT dbo.AñosPelícula ('Cocon', GetDate ())
```

Devolviendo 45. Para borrar la función creada haríamos:

```
DROP FUNCTION AñosPelícula
```

Las funciones escalares son muy similares a procedimientos almacenados con parámetros de salida, pero estas pueden ser utilizadas en consultas de selección y en la cláusula WHERE de las mismas.

Las funciones no pueden ejecutar sentencias INSERT o UPDATE.

## ii) Funciones en línea

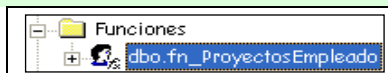
Las funciones en línea son las funciones que devuelven un conjunto de resultados correspondientes a la ejecución de una sentencia SELECT.

La sintaxis para una función de tabla en línea es la siguiente:

```
CREATE FUNCTION <Inline_Function_Name, sysname, FunctionName>  
(  
    -- Lista de parámetros  
    <@param1, sysname, @p1> <Data_Type_For_Param1, int>, ...  
)  
    RETURNS TABLE AS RETURN  
(  
    -- Sentencia Transact SQL  
)
```

El siguiente ejemplo muestra como crear una función en línea.

```
CREATE FUNCTION fn_ProyectosEmpleado (@Id smallint)
RETURNS TABLE AS RETURN
(
    SELECT Proyecto.*
    FROM Proyecto
        INNER JOIN Empleados
        ON Empleados.CódigoEm = Proyecto.CódigoEm
    WHERE Proyecto.CódigoEm = @Id
)
```



No podemos utilizar la cláusula ORDER BY en la sentencia de una función en línea. Las funciones en línea pueden utilizarse dentro de JOINS o QUERYS como si fueran una tabla normal.

```
SELECT *
FROM fn_ProyectosEmpleado (1)
```

	CódigoPr	PresupPr	Función	CódigoEm
1	1	100.0000	Proyecto número 1	1
2	5	500.0000	Proyecto número 5	1

Otro ejemplo de llamada a la función anterior sería el siguiente

```
SELECT DNIFa, NombreEm, SueldoEm, Jefe, PresupPr, Función
FROM Familiares
    INNER JOIN Empleados
    ON Familiares.CódigoEm = Empleados.CódigoEm
    INNER JOIN fn_ProyectosEmpleado (1) AS A
    ON Empleados.CódigoEm = A.Id
```

	DNIFa	NombreEm	SueldoEm	Jefe	PresupPr	Función
1	10000000A	Nombre del empleado 1	100.0000	NULL	100.0000	Proyecto número 1
2	10000000A	Nombre del empleado 1	100.0000	NULL	500.0000	Proyecto número 5



### iii) Funciones de tabla de multisentencias

Las funciones en línea de múltiples sentencias son similares a las funciones en línea excepto que el conjunto de resultados que devuelven puede estar compuesto por la ejecución de varias consultas SELECT.

Este tipo de función se usa en situaciones donde se requiere una mayor lógica de proceso.

La sintaxis para una función de tabla de multisentencias es la siguiente:

```
CREATE FUNCTION <Table_Function_Name, sysname, FunctionName>
(
    -- Lista de parámetros
    <@param1, sysname, @p1> <data_type_for_param1, , int>, ...
)
RETURNS
    -- variable de tipo tabla y su estructura
    <@Table_Variable_Name, sysname, @Table_Var> TABLE
(
    <Column_1, sysname, c1> <Data_Type_For_Column1, , int>,
    <Column_2, sysname, c2> <Data_Type_For_Column2, , int>
)
AS
BEGIN
    -- Sentencias que cargan de datos la tabla declarada
    RETURN
END
```

El siguiente ejemplo muestra el uso de una función de tabla de multisentencias que busca a un empleado poniendo el departamento y los proyectos en los que está involucrado.

```
CREATE FUNCTION fn_EmplDptoProy (@Id smallint)
    RETURNS @DatosSalida TABLE (
        -- Estructura de la tabla que devuelve la función.
        CódigoEm          smallint,
        NombreEm           varchar (50),
        NombreDe           varchar (50),
        SueldoEm           smallmoney,
        Jefe               smallint,
        PresupPr           smallmoney,
        Función             varchar (50)
    )
AS
```

```
BEGIN
-- Variables necesarias para la lógica de la función.
DECLARE    @CódigoEm smallint, @VarNombreEm varchar (50),
           @VarNombreDe varchar (50), @VarSueldoEm smallmoney,
           @VarJefe smallint, @VarPresupPr smallmoney,
           @VarFunción varchar (50)

-- Carga los datos del empleado solicitado
DECLARE CDatos CURSOR FOR
    SELECT Empleados.CódigoEm, NombreEm, NombreDe, SueldoEm,
           Jefe, PresupPr, Función
    FROM Empleados
    INNER JOIN Proyecto
    ON Empleados.CódigoEm = Proyecto.CódigoEm
    INNER JOIN Departamentos
    ON Empleados.CódigoDe =
        Departamentos.CódigoDe
    WHERE Empleados.CódigoEm = @Id

OPEN CDatos

-- Carga en CDatos los nombres de las variables
FETCH CDatos
    INTO @CódigoEm, @VarNombreEm, @VarNombreDe, @VarSueldoEm,
        @VarJefe, @VarPresupPr, @VarFunción

-- Recorremos el cursor
WHILE (@@FETCH_STATUS = 0)
BEGIN
    -- Cargamos en NombreDe el nombre del Departamento
    SELECT @VarNombreDe = NombreDe
    FROM Departamentos
    WHERE CódigoEm = @Id

    -- Cargamos VarPresupPr y VarFunción con el presupuesto y la función
    -- del proyecto respectivamente
    SELECT @VarPresupPr = PresupPr
    FROM Proyecto
    WHERE CódigoEm = @Id
    SELECT @VarFunción = Función
    FROM Proyecto
    WHERE CódigoEm = @Id
```

```

-- Insertamos los datos del Empleado en la variable de salida
INSERT INTO @DatosSalida
    (CódigoEm, NombreEm, NombreDe, SueldoEm,
     Jefe, PresupPr, Función)
VALUES
    (@Id, @VarNombreEm, @VarNombreDe, @VarSueldoEm,
     @VarJefe, @VarPresupPr, @VarFunción)

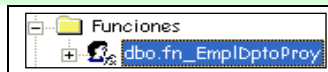
-- Vamos a la siguiente cuenta
FETCH Cdatos
    INTO @CódigoEm, @VarNombreEm, @VarNombreDe,
        @VarSueldoEm, @VarJefe, @VarPresupPr, @VarFunción

END

-- Cierra el cursor creado
CLOSE Cdatos
DEALLOCATE Cdatos

-- Retorna los valores cargados
RETURN
END

```



Para ejecutar la función:

```

SELECT *
FROM fn_EmplDptoProy (1)

```

	CódigoEm	NombreEm	NombreDe	SueldoEm	Jefe	PresupPr	Función
1	2	Nombre del empleado 2	Contabilidad	200.0000	1	200.0000	Proyecto número 2

## 10) Procedimientos almacenados

Un procedimiento es un programa dentro de la base de datos que ejecuta una acción o conjunto de acciones específicas. Un procedimiento tiene un nombre, un conjunto de parámetros (opcional) y un bloque de código.

En Transact SQL los procedimientos almacenados pueden devolver valores (numérico entero) o conjuntos de resultados.

Para crear un procedimiento almacenado debemos emplear la sentencia CREATE PROCEDURE.

```
CREATE PROCEDURE <NombreProcedimiento> [@param1 <tipo>], [@param2 <tipo>,...]  
AS  
-- Bloque de código
```

Para modificar un procedimiento almacenado debemos emplear la sentencia ALTER PROCEDURE.

```
ALTER PROCEDURE <NombreProcedimiento> [@param1 <tipo>], [@param2 <tipo>,...]  
AS  
-- Bloque de código
```

Para la ejecución un procedimiento almacenado debemos utilizar la sentencia EXEC. Cuando la ejecución del procedimiento almacenado es la primera instrucción del lote, podemos omitir el uso de EXEC. Por ejemplo para saber las conexiones que tiene nuestro servidor podemos hacer:

```
EXEC sp_who
```

El siguiente ejemplo muestra un procedimiento almacenado, denominado pa\_MuestraNombre que muestra el nombre de un empleado solicitado mediante su código sobre la tabla Empleados.

```
CREATE PROCEDURE pa_MuestraNombre  
    @Id smallint  
AS  
    SELECT NombreEm  
        FROM Empleados  
        WHERE CódigoEm = @Id
```

Para ejecutar este procedimiento haríamos:

```
EXEC pa_MuestraNombre 1
```

Borra el procedimiento almacenado

```
DROP PROCEDURE pa_MuestraNombre  
GO
```

El siguiente ejemplo muestra un procedimiento almacenado, denominado pa\_AddEmpleado que inserta un registro en la tabla Empleados.

```
CREATE PROCEDURE pa_AñadeSocio
    @DNIEm      char (9),
    @NombreEm   varchar (50),
    @DireccEm   varchar (50),
    @SueldoEm   smallmoney,
    @Jefe       smallint,
    @CódigoDe   smallint
AS
INSERT INTO Empleados (DNIEm, NombreEm, DireccEm, SueldoEm, Jefe, CódigoDe)
VALUES (@DNIEm, @NombreEm, @DireccEm, @SueldoEm, @Jefe, @CódigoDe)
GO
```

Y para ejecutar este procedimiento haríamos:

```
DECLARE @Dpto smallint
SET @Dpto = 1
EXEC pa_AñadeSocio '00000000A', 'Otro empleado', 'Con su dirección', 100, 1, @Dpto
-- Borra el procedimiento almacenado
DROP PROCEDURE pa_AñadeSocio
GO
```

Si queremos que los parámetros de un procedimiento almacenado sean de entrada-salida debemos especificarlo a través de la palabra clave OUTPUT, tanto en la definición del procedimiento como en la ejecución.

Si queremos obtener valores se recomienda utilizar parámetros de salida o funciones escalares.

El siguiente ejemplo muestra la definición de un procedimiento con parámetros de salida, es decir, que devuelve valores.

```
CREATE PROCEDURE pa_ObtenerSueldo
    @Id          smallint,
    @Sueldo      smallmoney OUTPUT
AS
BEGIN
    SELECT @Sueldo = SueldoEm
    FROM Empleados
    WHERE CódigoEm = @Id
END
GO
```

Y para ejecutar este procedimiento:

```
DECLARE @Sueldo smallmoney
EXEC pa_ObtenerSueldo 1, @Sueldo OUTPUT
IF @OtraVariableParaSueldo >100
    PRINT Cast(@OtraVariableParaSueldo AS varchar)
ELSE
    PRINT 'El valor es menor o igual a 100'
```

Y para borrar el procedimiento:

```
DROP PROCEDURE pa_ObtenerSueldo
GO
```

Por ejemplo para contar el número de ejemplares de una película sobre la base de datos Proyectalia.

```
CREATE PROCEDURE Pr_CuentaPelículas
    @Buscado varchar (50),
    @Número smallint OUTPUT
AS
    SELECT @Número = Count (*)
        FROM Películas
            INNER JOIN Ejemplares
                ON Películas.IdPelícula = Ejemplares.IdPelícula
        WHERE Películas.Título = @Buscado
GO
```

Para comprobación de su funcionamiento.

```
DECLARE @NúmeroPel int
SET @NúmeroPel=0
EXEC Pr_CuentaPelículas 'CoCon', @NúmeroPel OUTPUT
PRINT @NúmeroPel
```

Borra el procedimiento almacenado

```
DROP PROCEDURE Pr_CuentaPelículas
```

Otra característica muy interesante de los procedimientos almacenados en Transact SQL es que pueden devolver uno o varios conjuntos de resultados.

El siguiente ejemplo muestra un procedimiento almacenado que devuelve un conjunto de resultados.

```
CREATE PROCEDURE pa_ProyectosEmpleado @Id smallint
AS
BEGIN
    SELECT Empleados.CódigoEm, DNIEm, NombreEm, PresupPr, Función
        FROM Empleados
            INNER JOIN Proyecto
                ON Empleados.CódigoEm = Proyecto.CódigoEm
        WHERE Empleados.CódigoEm = @Id
        ORDER BY PresupPr DESC
END
GO
```

Para comprobar el correcto funcionamiento del procedimiento haremos:

```
EXEC pa_ProyectosEmpleado 1
GO
```

Y para borrar el procedimiento almacenado:

```
DROP PROCEDURE pa_ProyectosEmpleado
GO
```

Un procedimiento almacenado puede devolver valores numéricos enteros a través de la instrucción RETURN. Normalmente debemos utilizar los valores de retorno para determinar si la ejecución del procedimiento ha sido correcta o no.

```
CREATE PROCEDURE pa_ProyectoMenorCero @Id smallint
AS
BEGIN
    IF (SELECT PresupPr
        FROM Proyecto
        WHERE CódigoPr = @Id) <= 0
        RETURN 1
    ELSE
        RETURN 0
END
GO
```

Para ejecutar el procedimiento y obtener el valor devuelto.

```
DECLARE @ResultadoProcedimiento int
EXEC @ResultadoProcedimiento = pa_ProyectoMenorCero 1
PRINT @ResultadoProcedimiento
GO
```

Y para borrar el procedimiento almacenado:

```
DROP PROCEDURE pa_ProyectoMenorCero
GO
```

Un procedimiento almacenado al que le pasemos un código de una película y un nuevo estado de conservación. Si existe que lo cambie, y si no que nos diga que no existe.

```
CREATE PROCEDURE pa_CambiaEstadoConservación
    @IdEjemplar smallint,
    @NuevoEstCons varchar(50)
AS
    DECLARE @Salida smallint
    SET @Salida = 1
    IF EXISTS (SELECT *
        FROM Ejemplares
        WHERE IdEjemplar = @IdEjemplar)
        BEGIN
            UPDATE Ejemplares
                SET EstadoCons = @NuevoEstCons
                WHERE IdEjemplar = @IdEjemplar
            RETURN 0 -- Salida exitosa
        END
    ELSE
        BEGIN
            PRINT 'Ejemplar no coincidente. Error: 1'
            RETURN 1 -- Salida fallida
        END
GO
```

Y para borrar el procedimiento almacenado:

```
SELECT *
  FROM Ejemplares
 WHERE IdEjemplar = 1
DECLARE @RP int
EXEC @RP = pa_CambiaEstadoConservación 1, 'Malísimo'
PRINT @RP
SELECT *
  FROM Ejemplares
 WHERE IdEjemplar = 1
```

Y para borrar el procedimiento almacenado:

```
DROP PROCEDURE pa_CambiaEstadoConservación
```

Crear unprocedimiento almacenado al que se le pase una Fecha de alquiler y devuelva Nombre del Socio y Título de la películas alquiladas en esa fecha.

```
CREATE PROCEDURE pa_InfoFecha
  @Fecha smalldatetime
AS
  IF EXISTS (SELECT *
             FROM Alquila
             WHERE FechaAlqu = @Fecha)
  BEGIN
    SELECT NombreSoc, Título
      FROM Alquila
           INNER JOIN Socios
             ON Alquila.IdSoc = Socios.IdSoc
           INNER JOIN Películas
             ON Alquila.IdPelícula = Películas.IdPelícula
    WHERE FechaAlqu = @Fecha And FechaDev IS NULL
    RETURN 0    -- Salida exitosa
  END
ELSE
  BEGIN
    PRINT 'Fecha sin alquileres. Error: 1'
    RETURN 1    -- Salida fallida
  END
GO
```

Y para borrar el procedimiento almacenado:

```
SELECT *
  FROM ALQUILA

DECLARE @RP int
EXEC @RP = pa_InfoFecha '12/11/2008'
PRINT @RP
```

Y para borrar el procedimiento almacenado:

```
DROP PROCEDURE pa_InfoFecha
```



## 11) Disparadores

Un trigger (o desencadenador, o disparador) es una clase especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento en el servidor de bases de datos.

SQL Server proporciona los siguientes tipos de triggers:

- **Trigger DML**, se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista.
- **Trigger DDL**, se ejecutan en respuesta a una variedad de eventos de lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a instrucciones CREATE, ALTER y DROP de Transact-SQL, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL.

### a) Trigger DML.

La sintaxis general de un trigger es la siguiente:

```
CREATE TRIGGER Nombre
  ON <Tabla>
  FOR / AFTER / BEFORE <Acción, INSERT, DELETE, UPDATE>
AS
BEGIN
  -- Se añade para que no aparezcan resultados indeseados en el SELECT
  SET NOCOUNT ON
  -- Acciones del disparador
  SET NOCOUNT OFF
END
```

El siguiente ejemplo, muestra un mensaje al modificar la tabla de empleados.

```
CREATE TRIGGER TR_ModificaEmpleados
  ON Empleados
  FOR UPDATE
AS
PRINT 'Usted ha modificando la tabla empleados'
```

Para comprobar el funcionamiento del disparador anterior podemos modificar cualquier registro de la tabla empleados.

```
UPDATE empleados
  SET CódigoDe = 1
  WHERE CódigoDe = 1
```

Antes de examinar detenidamente cualquier ejemplo es necesario conocer las tablas INSERTED y DELETED. Las instrucciones de triggers DML utilizan dos tablas especiales denominadas INSERTED y DELETED. SQL Server crea y administra automáticamente ambas tablas. La estructura de las tablas INSERTED y DELETED es la misma que tiene la tabla que ha desencadenado la ejecución del trigger.

La primera tabla (INSERTED) sólo está disponible en las operaciones INSERT y UPDATE, y en ella están los valores resultantes después de la inserción o actualización. Es decir, los datos insertados. INSERTED estará vacía en una operación DELETE.

En la segunda (DELETED), disponible en las operaciones UPDATE y DELETE, están los valores anteriores a la ejecución de la actualización o borrado. Es decir, los datos que serán borrados. DELETED estará vacía en una operación INSERT.

¿No existe una tabla UPDATED? No, hacer una actualización es lo mismo que borrar (DELETED) e insertar los nuevos (INSERTED). La sentencia UPDATE es la única en la que INSERTED y DELETED tienen datos simultáneamente.

Dos consideraciones:

- No se puede modificar directamente los datos de estas tablas.
- El trigger se ejecutará aunque la instrucción DML (UPDATE, INSERT o DELETE) no haya afectado a ninguna fila. En este caso INSERTED y DELETED devolverán un conjunto de datos vacío.

El siguiente ejemplo, crear un disparador que añada el título, el identificador de ejemplar y fecha de alquiler en una tabla previamente creada (tiene que existir llamada ControlAlquileres), al insertar registros en la tabla Alquila.

```
-- Crea la tabla de histórico
CREATE TABLE ControlAlquileres (
    NúmeroControl smallint          IDENTITY (1, 1)
                                PRIMARY KEY,
    Título          varchar (50)    NOT NULL,
    IdEjemplar      smallint         NOT NULL,
    FechaAlquiler  smalldatetime    DEFAULT GetDate()
)
GO
```

Para crea el disparador:

```
CREATE TRIGGER tr_AñadeHistório
ON Alquila
FOR INSERT
AS
BEGIN
    -- Inserto los datos en la tabla nueva
    INSERT INTO ControlAlquileres (Título, IdEjemplar)
        SELECT Películas.Título, IdEjemplar
        FROM Inserted
        INNER JOIN Películas
        ON Inserted.IdPelícula = Películas.IdPelícula
END
```

La siguiente instrucción provocará que el trigger se ejecute:

```
INSERT INTO Alquila (FechaAlqu, IdEjemplar, IdPelícula, IdSoc)
VALUES (GetDate(), 1, 1, 1)
```

Y podremos comprobar como la tabla ControlAlquileres vá creciendo mediante las siguientes órdenes:

```
SELECT *
FROM ControlAlquileres
```

	NúmeroControl	Título	IdEjemplar	FechaAlquiler
1	1	La bala que dobló la esquina	1	2008-12-13 08:20:00

Podemos activar y desactivar Triggers a través de las siguientes instrucciones:

```
-- Desactiva el disparador tr_AñadeHistórico de la tabla Alquila
ALTER TABLE Alquila DISABLE TRIGGER tr_AñadeHistórico
GO
-- Activa el disparador tr_AñadeHistórico de la tabla alquila
ALTER TABLE Alquila ENABLE TRIGGER tr_AñadeHistórico
GO
-- Desactiva todos los disparadores de la tabla Alquila
ALTER TABLE Alquila DISABLE TRIGGER ALL
GO
-- Activa todos los disparadores de la tabla Alquila
ALTER TABLE Alquila ENABLE TRIGGER ALL
GO
-- Borra el disparador
DROP TRIGGER tr_AñadeHistórico
GO
```

Los trigger están dentro de la transacción original (INSERT, DELETE o UPDATE) por lo cual si dentro de nuestro trigger hacemos un ROLLBACK TRAN, no solo estaremos echando atrás nuestro trigger sino también toda la transacción; en otras palabras si en un trigger ponemos un ROLLBACK TRAN, la transacción de INSERT, DELETE o UPDATE volverá toda hacia atrás.

Otro ejemplo para crear un disparador para comprobar que un Socio no pueda alquilar más de 3 ejemplares.

```
CREATE TRIGGER tr_MaximoAlquileresSocio
ON Alquila
FOR INSERT
AS
BEGIN
    -- Calculo el número de Alquileres del socio
    DECLARE @NúmeroAlquileres smallint
    SELECT @NúmeroAlquileres = Count (*)
        FROM Alquila
        WHERE FechaDev IS NULL And
            IdSoc = (SELECT IdSoc
                    FROM Inserted)
    IF @NúmeroAlquileres>3
    BEGIN
        PRINT 'Ha superado el número máximo de alquileres permitido (3)'
        ROLLBACK
    END
END
```

El socio 2 ya tenía un alquiler no devuelto realizado. Para probar el funcionamiento del disparador tendremos que hacer dos inserciones en la tabla de alquileres. El primero nos dá como resultado que la fila ha sido añadida:


```
INSERT INTO Alquiler (FechaAlqu, IdEjemplar, IdPelícula, IdSoc)
VALUES (GetDate(), 2, 1, 2)
```



Cuando intentamos insertar el tercer alquiler:

```
INSERT INTO Alquiler (FechaAlqu, IdEjemplar, IdPelícula, IdSoc)
VALUES (GetDate(), 3, 1, 2)
```

Se nos advierte:



Y deshace los cambios solicitados (inserción del registro), con lo que el registro no es añadido a la base de datos.

Otro ejemplo podría ser aquel que borre las filas de la tabla ControlAlquileres cuando el cliente devuelva una película, pudiendo especificar a que columnas de la tabla debe afectar el trigger.

```
CREATE TRIGGER tr_DevuelvePelícula
ON Alquiler
AFTER UPDATE
AS
BEGIN
    IF UPDATE (FechaDev)      -- Sólo si se actualiza la fecha de devolución
    BEGIN
        -- Inicializa la variable necesaria
        DECLARE @IdEjemplar smallint
        SELECT @IdEjemplar = IdEjemplar
        FROM Inserted
        -- Borro la línea de ControlAlquileres
        DELETE
        FROM ControlAlquileres
        WHERE IdEjemplar = @IdEjemplar
    END
END
```

Para probar el funcionamiento del disparador, devuelvo una película, y la línea afectada de la tabla ControlAlquileres desaparecerá automáticamente.

```
UPDATE Alquiler
SET FechaDev = GetDate()
WHERE IdSoc = 1 And
      IdPelícula = 1 And
      IdEjemplar = 3
```

**b) Trigger DDL (Sólo para versiones superiores a la del 2005)**

La sintaxis general de un trigger es la siguiente.

```
CREATE TRIGGER Nombre
  ON DATABASE
  FOR / AFTER / BEFORE <Acción, DROP_TABLE, ALTER_TABLE>
AS
BEGIN
  -- Acciones del disparador
END
```

La siguiente instrucción impide que se ejecuten sentencias DROP TABLE y ALTER TABLE en la base de datos.

```
CREATE TRIGGER tr_Seguridad
  ON DATABASE
  FOR DROP_TABLE, ALTER_TABLE
AS
BEGIN
  RAISERROR ('No está permitido borrar ni modificar tablas!' , 16, 1)
  ROLLBACK TRANSACTION
END
```

## 12) Flujos de control

### a) La sentencia IF ... ELSE

Esta sentencia permite evaluar una expresión y provocar la entrada o ejecución de un bloque u otro, dependiendo de que la condición sea verdadera o falsa.

La sintaxis general de IF es:

```
IF (<expresión>)  
  BEGIN  
    ...  
  END  
ELSE IF (<expresión>)  
  BEGIN  
    ...  
  END  
ELSE  
  BEGIN  
    ...  
  END
```

El siguiente código muestra un ejemplo de una función IF con ELSE.

```
IF (SELECT Count (*)  
    FROM Empleados) = 0  
  BEGIN  
    Print 'No hay empleados en la tabla'  
  END  
ELSE  
  BEGIN  
    Print 'Ya existen empleados en la tabla'  
  END
```

### b) La sentencia CASE

Una variante a la instrucción IF..ELSE es la sentencia CASE, muy útil sobre todo cuando tenemos muchos IF anidados.

La sintaxis general de CASE es:

```
CASE <expresión>  
  WHEN <valor_ expresión > THEN <valor_devuelto>  
  WHEN <valor_ expresión > THEN <valor_devuelto>  
  ELSE <valor_devuelto>                -- Valor por defecto  
END
```

Por ejemplo:

```
DECLARE @Valor int
SELECT @Valor = Count (*)
                FROM Empleados
Print CASE @Valor
      WHEN '0' THEN
        'Ho hay empleados en la tabla'
      WHEN '1' THEN
        'Existe un empleado en la tabla'
      ELSE
        'Existen más de un empleado en la tabla'
END
```

Otro ejemplo:

```
DECLARE @Completo varchar (100), @Abreviado varchar (3)
SET @Abreviado = 'ITV'
SET @Completo = (CASE @Abreviado
                  WHEN 'ITV' THEN 'www.ItvSoft.net'
                  WHEN 'FPC' THEN 'www.ItvSoft.es'
                  ELSE 'www.google.es'
                  END)
PRINT @Web
```

Otro aspecto muy interesante de CASE es que permite el uso de subconsultas.

```
DECLARE @Completo varchar (100), @Abreviado varchar (3)
SET @Abreviado = ITV
SET @Completo = CASE
                  WHEN @Abreviado = 'ITV' THEN (SELECT NombreEm
                                                  FROM Empleados
                                                  WHERE CódigoEm=1)
                  WHEN @Abreviado = 'FPC' THEN (SELECT NombreEm
                                                  FROM Empleados
                                                  WHERE CódigoEm=2)
                  ELSE 'Ninguno'
                  END
PRINT @Completo
```

Otro ejemplo en el que mostramos un campo al que le añadimos ' (ES)' y otra columna que indica si es Actor o Actriz

```
SELECT NombreAc + ' (ES)' AS 'Añadido', Tipo = CASE SexoAc
                                WHEN 'Hombre'      THEN 'Actor'
                                WHEN 'Mujer'        THEN 'Actriz'
                                END
FROM Actores
WHERE NacionaliAc LIKE ('Españ%')
```

	Añadido	Tipo
1	Antonia Banderas (ES)	Actor
2	Patricia Conde (ES)	Actriz
3	Lucía la Piedra (ES)	Actriz
4	Pilar Rubio (ES)	Actriz
5	Jorge Javier Vazquez (ES)	Actor

### c) La sentencia WHILE

El bucle WHILE se repite mientras expresion se evalúe como verdadero.

Es el único tipo de bucle del que dispone Transact SQL.

```
WHILE <expresion>
BEGIN
    ...
END
```

Un ejemplo del bucle WHILE.

```
DECLARE @Contador int
SET @Contador = 0
WHILE (@Contador < 10)
BEGIN
    SET @Contador = @Contador + 1
    PRINT 'Iteracion del bucle ' + Cast (@Contador AS varchar)
END
```

Iteracion del bucle 1
Iteracion del bucle 2
Iteracion del bucle 3
Iteracion del bucle 4
Iteracion del bucle 5
Iteracion del bucle 6
Iteracion del bucle 7
Iteracion del bucle 8
Iteracion del bucle 9
Iteracion del bucle 10



Podemos pasar a la siguiente iteración del bucle utilizando CONTINUE.

```
DECLARE @Contador int
SET @Contador = 0
WHILE (@Contador < 10)
BEGIN
    SET @Contador = @Contador + 1
    IF (@Contador % 2 = 0)
        CONTINUE
    PRINT 'Iteracion del bucle ' + Cast (@contador AS varchar)
END
```

```
Iteracion del bucle 1
Iteracion del bucle 3
Iteracion del bucle 5
Iteracion del bucle 7
Iteracion del bucle 9
```

El bucle se dejará de repetir con la instrucción BREAK.

```
DECLARE @Contador int
SET @Contador = 0
WHILE (1 = 1)
BEGIN
    SET @Contador = @Contador + 1
    IF (@Contador % 5 = 0)
        BREAK
    PRINT 'Iteracion del bucle ' + Cast (@Contador AS varchar)
END
```

```
Iteracion del bucle 1
Iteracion del bucle 2
Iteracion del bucle 3
Iteracion del bucle 4
```

También podemos utilizar el bucle WHILE conjuntamente con subconsultas.

```
DECLARE @Contador int
SET @Contador = 0
DECLARE @PresupuestoMáximo smallmoney, @PresupuestoProyecto smallmoney
SET @PresupuestoMáximo = (SELECT Max(PresupPr)
                        FROM Proyecto)
WHILE EXISTS (SELECT *
              FROM Proyecto
              WHERE PresupPr <> @PresupuestoMáximo)
    -- La subconsulta se ejecuta una vez por cada proyecto
BEGIN
    SET @Contador = @Contador + 1
    UPDATE Proyecto
        SET PresupPr = @PresupuestoMáximo
    PRINT 'Presupuestos actualizados a: ' +
        Cast (@PresupuestoMáximo AS varchar)
END
```

```
(5 filas afectadas)
Presupuestos actualizados a: 500.00
```