STRUCTURED QUERY LANGUAGE - SQL

1. INTRODUCCIÓN

En este capítulo se introducirán conceptos básicos del lenguaje de consulta estructurado (SQL, Structured Query Language) y TransacT-SQL (T-SQL) y se mostrarán las diferencias entre los dos lenguajes. Este capítulo explica el lenguaje de definición de datos (DDL, Data Definition Language) y el lenguaje de tratamiento de datos (DML, Data Manipulation Language) y se incluyen ejemplos de cada uno. Además se aprenderá a utilizar las diferentes utilidades de SQL Server (incluyendo T-SQL de línea de comandos y el Analizador de consultas SQL) para crear y administrar objetos base de datos. También se aprenderá a crear secuencias dé comandos que contengan instrucciones T-SQL.

SQL

SQL es un lenguaje de consulta y programación de bases de datos utilizado para acceder a los datos y para consultar, actualizar y gestionar sistemas de bases de datos relacionales. Tanto ANSI (American National Standards Institute, Instituto nacional de normalización americano) como ISO (Internacional Organization for Standardization, Organización internacional para la estandarización) han definido estándares para SQL. ANSI es una organización de grupos industriales y de negocios que desarrollan estándares de comunicación y negocio para los Estados Unidos. ANSI también es un miembro de ISO y de IEC (Intemational Electrotechnical Commission, Comisión electrotécnica internacional). ANSI publica estándares para EEUU que se corresponden con los estándares internacionales. En 1992, ISO e IEC publicaron un estándar para SQL denominado SQL-92. ANSI publicó un estándar correspondiente, ANSI SQL-92, en EEUU. ANSI SQL-92 se conoce algunas veces como ANSI SQL. Aunque bases de datos relacionales diferentes utilicen versiones ligeramente diferentes de SQL, la mayoría cumple con el estándar ANSI SQL. SQL Server utiliza el superconjunto de ANSI SQL-92 conocido como T-SQL, el cual se ajusta al estándar, SQL 92 definido por ANSI.

El lenguaje SQL contiene instrucciones que se ajustan las dos principales categorías de programación: DDL y DML. Se verán estas categorías de lenguaje en las siguientes secciones.

DDL

DDL se utiliza para definir y administrar objetos bases de datos tales como bases de datos, tablas y vistas. Las instrucciones DDL usualmente incluyen instrucciones CREATE, ALTER y DROP para cada objeto. Por ejemplo, las instrucciones CREATE TABLE, ALTER TABLE y DROP TABLE se utilizan para crear una tabla, modificar sus propiedades (agregar o borrar columnas, por ejemplo) y eliminar una tabla, respectivamente.

DML

DML se utiliza para manipular los datos contenidos en los objetos base de datos. Para ello se utilizan instrucciones tales como INSERT, SELECT, UPDATE y DELETE. Estas instrucciones permiten seleccionar filas de datos mediante la realización de consultas, insertar nuevas filas de datos, modificar las filas de datos existentes y borrar filas de datos no deseadas, respectivamente. Esta sección proporciona ejemplos básicos de cada una de estas instrucciones.

T-SQL

SQL es una mejora del lenguaje de programación SQL estándar. Es el lenguaje principal utilizado para comunicaciones entre aplicaciones y SQL Server. T-SQL proporciona las posibilidades DDL y DML de SQL estándar además de funciones extendidas, procedimientos almacenados del sistema y con construcciones de programación (tales como IF y WHILE) con el fin de permitir mayor flexibilidad en la programación. Las capacidades de T-SQL continúan creciendo con las versiones nuevas de SQL Server.

2. DEFINICIÓN DE BASES DE DATOS

CREATE DATABASE

Crea una nueva base de datos y los ficheros usados para almacenar la base de datos o crea una nueva base de datos a partir de los ficheros de otra base de datos previamente creada.

```
CREATE DATABASE database_name
[ ON
[ < filespec > [ ,...n ] ]
[ , < filegroup > [ ,...n ] ]
]
[ LOG ON { < filespec > [ ,...n ] } ]
[ COLLATE collation_name ]
[ FOR LOAD | FOR ATTACH ]
< filespec > ::=
[ PRIMARY ]
( [ NAME = logical_file_name , ]
FILENAME = 'os_file_name'
[ , SIZE = size ]
[ , MAXSIZE = { max_size | UNLIMITED } ]
[ , FILEGROWTH = growth_increment ] ) [ ,...n ]
< filegroup > ::=
FILEGROUP filegroup_name < filespec > [ ,...n ]
```

Argumentos

database_name
Es el nombre de la base de datos. El nombre de la base de datos debe ser único en la base de datos y contener un máximo de 128 caracteres.

ON
A continuación se especifican los ficheros de disco usados para almacenar los datos de la base de datos. Esta palabra clave va seguida de una lista de <filespec> (especificaciones de ficheros) que definen los ficheros de datos para el grupo primario. Esta lista puede opcionalmente ir seguida de una lista de <filegroup> (grupos de ficheros) que define los grupos de ficheros y sus ficheros.

LOG ON
Especifica que ficheros de discos que usan para el registro de transacciones (log) de la base de datos. Esta palabra clave va seguida de una lista de <filespec> (especificaciones de ficheros) que definen los ficheros de registro de transacciones. Si no se especifica que crea un solo fichero de log con un nombre por defecto generado pro el sistema y con un tamaño del 25% del tamaño de los ficheros de datos.

FOR LOAD
Esta opción se usa por compatibilidad con versiones anteriores. La base de datos se crea con la opción de dbo use only.

FOR ATTACH
Esta opción se usa para crear bases de datos usando ficheros de datos ya creados anteriormente con otra base de datos.

collation_name
Especifica el valor por defecto de la tabla de carácteres para la base de datos. Si no se especifica SQL Server pone el que tenga dicha instancia del servidor de SQL Server.

Ejemplos

- **Crear una base de datos con todas las opciones por defecto**

```
CREATE DATABASE mytest
```

- **Crear una base de datos con un fichero de datos en el grupo Primario**

```
USE master
GO
CREATE DATABASE Products
ON
(    NAME = prods_dat,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\prods.mdf',
     SIZE = 4, MAXSIZE = 10, FILEGROWTH = 1 )
GO
```

- **Crear una base de datos con un fichero de datos y uno de registro**

```
USE master
GO
CREATE DATABASE  Sales
ON
(    NAME = Sales_dat,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\saledat.mdf',
     SIZE = 10, MAXSIZE = 50, FILEGROWTH = 5 )
LOG ON
```

```
(    NAME = 'Sales_log',
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\salelog.ldf',
     SIZE = 5MB, MAXSIZE = 25MB,  FILEGROWTH = 5MB )
GO
```

- **Crear una base de datos con varios ficheros de datos y varios de registro**

```
USE master
GO
CREATE DATABASE Archive
ON PRIMARY
(    NAME = Arch1,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\archdat1.mdf',
     SIZE = 100MB, MAXSIZE = 200, FILEGROWTH = 20),
(    NAME = Arch2,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\archdat2.ndf',
     SIZE = 100MB, MAXSIZE = 200, FILEGROWTH = 20),
 (   NAME = Arch3,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\archdat3.ndf',
     SIZE = 100MB, MAXSIZE = 200, FILEGROWTH = 20)
LOG ON
(    NAME = Archlog1,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\archlog1.ldf',
     SIZE = 100MB, MAXSIZE = 200, FILEGROWTH = 20),
(    NAME = Archlog2,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\archlog2.ldf',
     SIZE = 100MB, MAXSIZE = 200, FILEGROWTH = 20)
GO
```

- **Crear una base de datos con varios grupos de ficheros**

```
CREATE DATABASE Sales
ON PRIMARY
(    NAME = SPri1_dat,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\SPri1dat.mdf',
      SIZE = 10, MAXSIZE = 50, FILEGROWTH = 15% ),
(    NAME = SPri2_dat,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\SPri2dt.ndf',
     SIZE = 10, MAXSIZE = 50, FILEGROWTH = 15% ),
FILEGROUP SalesGroup1
(    NAME = SGrp1Fi1_dat,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\SG1Fi1dt.ndf',
     SIZE = 10, MAXSIZE = 50, FILEGROWTH = 5 ),
(    NAME = SGrp1Fi2_dat,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\SG1Fi2dt.ndf',
     SIZE = 10, MAXSIZE = 50, FILEGROWTH = 5 ),
FILEGROUP SalesGroup2
(    NAME = SGrp2Fi1_dat,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\SG2Fi1dt.ndf',
     SIZE = 10, MAXSIZE = 50, FILEGROWTH = 5 ),
 (   NAME = SGrp2Fi2_dat,
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\SG2Fi2dt.ndf',
     SIZE = 10, MAXSIZE = 50, FILEGROWTH = 5 )
LOG ON
(    NAME = 'Sales_log',
     FILENAME = 'c:\program files\microsoft sql server\mssql\data\salelog.ldf',
      SIZE = 5MB, MAXSIZE = 25MB, FILEGROWTH = 5MB )
GO
```

3. MANIPULACIÓN DE BASES DE DATOS

DROP DATABASE

Elimina una o más bases de datos SQL Server. Borrar una base de datos elimina la base de datos y todos los ficheros usados en esa base de datos.

DROP DATABASE database_name [,...n]

Argumentos

database_name
Especifica el nombre de la base de datos. Las bases de datos del sistema (**msdb**, **master**, **model**, **tempdb**) no se deben borrar.

Ejemplos

- **Borrar una sola base de datos**

  DROP DATABASE publishing

- **Borrar varias bases de datos**

  DROP DATABASE pubs, newpubs

ALTER DATABASE

Añade o elimina ficheros o grupos de ficheros de una base de datos. Se puede usar también para modificar las propiedades de ficheros y grupos de ficheros.

```
ALTER DATABASE database
{ ADD FILE < filespec > [ ,...n ] [ TO FILEGROUP filegroup_name ]
| ADD LOG FILE < filespec > [ ,...n ]
| REMOVE FILE logical_file_name
| ADD FILEGROUP filegroup_name
| REMOVE FILEGROUP filegroup_name
| MODIFY FILE < filespec >
| MODIFY NAME = new_dbname
| MODIFY FILEGROUP filegroup_name {filegroup_property | NAME = new_filegroup_name }
| SET < optionspec > [ ,...n ] [ WITH < termination > ]
| COLLATE < collation_name >
}
```

Ejemplos

- **Añadir un fichero a la base de datos**

```
USE master
GO

CREATE DATABASE Test1 ON
(
 NAME = Test1dat1,
 FILENAME = 'c:\Program Files\Microsoft SQL Server\MSSQL\Data\t1dat1.ndf',
```

```
 SIZE = 5MB, MAXSIZE = 100MB, FILEGROWTH = 5MB
)
GO

ALTER DATABASE Test1
ADD FILE
(
 NAME = Test1dat2,
 FILENAME = 'c:\Program Files\Microsoft SQL Server\MSSQL\Data\t1dat2.ndf',
 SIZE = 5MB, MAXSIZE = 100MB, FILEGROWTH = 5MB
)
GO
```

- **Añadir un grupo de ficheros a la base de datos**

```
USE master
GO

ALTER DATABASE Test1
ADD FILEGROUP Test1FG1
GO

ALTER DATABASE Test1
ADD FILE
( NAME = test1dat3,
  FILENAME = 'c:\Program Files\Microsoft SQL Server\MSSQL\Data\t1dat3.ndf',
  SIZE = 5MB,
  MAXSIZE = 100MB,
  FILEGROWTH = 5MB),
( NAME = test1dat4,
  FILENAME = 'c:\Program Files\Microsoft SQL Server\MSSQL\Data\t1dat4.ndf',
  SIZE = 5MB,
  MAXSIZE = 100MB,
  FILEGROWTH = 5MB)
TO FILEGROUP Test1FG1

ALTER DATABASE Test1
MODIFY FILEGROUP Test1FG1 DEFAULT
GO
```

- **Añadir ficheros de log a la base de datos**

```
USE master
GO

ALTER DATABASE Test1
ADD LOG FILE
( NAME = test1log2,
  FILENAME = 'c:\Program Files\Microsoft SQL Server\MSSQL\Data\test2log.ldf',
  SIZE = 5MB, MAXSIZE = 100MB, FILEGROWTH = 5MB),
( NAME = test1log3,
  FILENAME = 'c:\Program Files\Microsoft SQL Server\MSSQL\Data\test3log.ldf',
  SIZE = 5MB, MAXSIZE = 100MB, FILEGROWTH = 5MB)
GO
```

- **Eliminar ficheros de la base de datos**

```
USE master
GO
```

```
ALTER DATABASE Test1
REMOVE FILE test1dat4
GO
```

- **Modificar un fichero**

```
USE master
GO
ALTER DATABASE Test1
MODIFY FILE
  (NAME = test1dat3,
  SIZE = 20MB)
GO
```

- **Convertir un grupo de ficheros en grupo por defecto**

```
USE master
GO
ALTER DATABASE MyDatabase
MODIFY FILEGROUP [PRIMARY] DEFAULT
GO
```

USE DATABASE

Cambia la base de datos en uso por la especificada.

```
USE { database }
```

4. DEFINICIÓN DE TABLAS

CREATE TABLE

Permite crear una nueva tabla.

```
CREATE TABLE
[ database_name.[ owner ] . | owner. ] table_name
( { < column_definition >
  | column_name AS computed_column_expression
  | < table_constraint > ::= [ CONSTRAINT constraint_name ] }
  | [ { PRIMARY KEY | UNIQUE } [ ,...n ]
)

[ ON { filegroup | DEFAULT } ]
[ TEXTIMAGE_ON { filegroup | DEFAULT } ]

< column_definition > ::= { column_name data_type }
  [ COLLATE < collation_name > ]
  [ [ DEFAULT constant_expression ]
    | [ IDENTITY [ ( seed , increment ) [ NOT FOR REPLICATION ] ] ]
  ]
 [ ROWGUIDCOL]
 [ < column_constraint > ] [ ...n ]

< column_constraint > ::= [ CONSTRAINT constraint_name ]
 { [ NULL | NOT NULL ]
   | [ { PRIMARY KEY | UNIQUE }
       [ CLUSTERED | NONCLUSTERED ]
       [ WITH FILLFACTOR = fillfactor ]
       [ON {filegroup | DEFAULT} ] ]
  ]
  | [ [ FOREIGN KEY ]
      REFERENCES ref_table [ ( ref_column ) ]
      [ ON DELETE { CASCADE | NO ACTION } ]
      [ ON UPDATE { CASCADE | NO ACTION } ]
      [ NOT FOR REPLICATION ]
  ]
  | CHECK [ NOT FOR REPLICATION ]
    ( logical_expression )
  }

< table_constraint > ::= [ CONSTRAINT constraint_name ]
  { [ { PRIMARY KEY | UNIQUE }
     [ CLUSTERED | NONCLUSTERED ]
     { ( column [ ASC | DESC ] [ ,...n ] ) }
     [ WITH FILLFACTOR = fillfactor ]
     [ ON { filegroup | DEFAULT } ]
  ]
  | FOREIGN KEY
     [ ( column [ ,...n ] ) ]
    REFERENCES ref_table [ ( ref_column [ ,...n ] ) ]
    [ ON DELETE { CASCADE | NO ACTION } ]
    [ ON UPDATE { CASCADE | NO ACTION } ]
    [ NOT FOR REPLICATION ]
    | CHECK [ NOT FOR REPLICATION ]
```

(*search_conditions* ***)***
}

Argumentos

database_name
Es el nombre de la base de datos en la que se crea la tabla. Si no se especifica ninguna base de datos se crea en la base de datos que esté en uso actualmente.

owner
Es el nombre del usuario propietario de la base de datos, que debe ser un identificador de usuario valido en la base de datos donde se intenta crear la tabla. Si no se especifica ningun usuarios se utilizará el utilizado para acceder al servidor de base de datos actualmente.

table_name
Es el nombre de la tabla. La combinación *owner.table_name* debe ser única en la base de datos y debe contener un máximo de 128 caractéres.

column_name
Nombre de una columna. El nombre de cada columna debe ser único en la tabla y debe seguir las normas de identificadores de SQL Server. Toda columna debe tener un nombre, a excepción de la columna marcada con el tipo *timestamp* que puede no llevarlo, en cuyo caso el sistema le pone el nombre por defecto timestamp.

computed_column_expression
Se utilizará si los datos almacenados en la columna son el resultado de realizar alguna operación sobre otra u otras columnas. Una expresión valida puede ser: el nombre de una columna, uan constante, una fucnión una variable o una combinación de las anteriores. La expresión no puede ser una subquery.

ON {*filegroup* | DEFAULT}
Especifica el grupo de ficheros en el que se almacena la tabla. Si no se especifica ningún grupo de ficheros la tabla se almacenará en el grupo de ficheros por defecto.

TEXTIMAGE_ON
Indica el grupo de ficheros en el que se almacena las columnas de los tipos text, ntext y image. Si no hay columnas de estos tipos en las tablas no se puede incluir esta opción. Si no se especifica esta opción, las columnas de este tipo se guardan en el mismo grupo de ficheros que la tabla.

data_type
Especifica el tipo de datos de la columna. Se aceptan tipos de datos del sistema o definidos por el usuario. Si el tipo de datos es de los definidos por el usuario al crear la tabla no se puede cambiar la longitud definida al crear el tipo, aunque si se puede cambiar su condición de NULL / NOT NULL.

DEFAULT
Especifica el valor por defecto asignado a esa columna cuando no se indica valor para ella en una sentencia de Insert.

constant_expression
Puede ser una constante, NULL o una función del sistema.
IDENTITY
Indica que la nueva columna es una columna de tipo Identity (Autonumérico). Esta propiedad se puede añadir a columnas del tipo tinyint, smallint, int, bigint, decimal(p,0), numeric(p,0). Solo una columna por tabla puede ser de este tipo. Si una columan tiene esta propiedad se debe especificar la semilla y el incremento, es decir, el primer valor de la columna y el incremento entre valores consecutivos. Si no se especifican estos parámetros se toma por defecto como semilla 1 y como incremento 1.

seed
Es el valor para la primera fila de una columna que tenga la propiedad Identity.

increment
Es el incremento con respecto al valor de la fila anterior para una columna con la propiedad Identity.

ROWGUIDCOL
Indica que esta columna es del tipo RowGuidCol. Solo puede haber una por tabla.

collation_name
Especifica la tabla de caracteres para dicha columna. Solo se puede aplicar a columnas de los tipos char, varchar, text, nchar, nvarchar, ntext. Si no se especifica, a la columna se le asigna la tabla de caracteres del tipo de datos que tenga la columna, si este es un tipo de datos definido por el usuario, o la tabla de caracteres de la base de datos donde se encuentre la tabla.

CONSTRAINT
Indica que la columna tiene alguna de las siguientes restricciones: PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY, CHECK

constraint_name
Nombre de la restricción. El nombre de la restricción debe ser único en la base de datos.

NULL | NOT NULL
Determina si se permiten o no valores nulos en una columna.

PRIMARY KEY
Determina si una columna o columnas son clave primaria. Solo se puede crear una restricción de este tipo por tabla, y asegura que estas columnas cumplen la regla de restricción de integridad de la entidad.

UNIQUE
Indica que esta columna forma parte de un índice unico, es decir, un índice en el que no se permiten valores duplicados. Esto asegura que estas columnas cumplen la regla de restricción de integridad de la entidad.

CLUSTERED | NONCLUSTERED
Indica que las columnas forman parte de un índice clusteres o nonclusteres.

FOREIGN KEY**...**REFERENCES
Indica que esta columna o columnas forman parte de una clave foránea, lo que asegura que estas columnas cumplen la regla de restricción de integridad referencia.

ref_table
Es el nombre de la tabla referenciada por una restricción de clave primaria.

(*ref_column*[**,**...*n*]**)**
Es el nombre de la columna o columnas referenciadas por una restricción de clave primaria.

ON DELETE {CASCADE | NO ACTION}
Especifica que acción se realiza sobre una fila de esta tabla, si se borra la fila con la que esta relacionada en otra tabla. Por defecto no se realiza ninguna acción.

ON UPDATE {CASCADE | NO ACTION}
Lo mismo que la regla anterior pero para el caso de actualizaciones.

CHECK
Limita los posibles valores que se pueden poner en una columna.

logical_expression
Expresión lógica que tiene que ser verdadera o falsa.

column
Nombre de la columna o columnas usadas en la restricción

[ASC | DESC]
Especifica el orden en que las columnas participan en la restricción de la tabla. El valor por defecto es ascendente.

Consideraciones

- SQL Server puede incluir hasta 2 billones de tablas en cada base de datos y 1024 columnas en cada tabla como máximo. El tamaño máximo de cada fila es de 8060 bytes.

- Se puede crear como máximo 1 indice de tipo clusteres y 249 índices de tipo nonclustered incluyendose en estos los generados por las restricciones de clave primaria y foráneas, y por restricciones UNIQUE.

- Una tabla sólo puede contener una restricción de clave primaria.

- Todas las columnas que forman parte de la clave primaria deben estar definidas con NOT NULL.

- Cada columna definida con una restricción UNIQUE genera un índice

- Las restricciones de clave foránea, solo puede referenciar a tablas situadas en la misma base de datos del mismo servidor. Estas restricciones también pueden hacer referencias a campos en la misma tabla (auto-referencia).

- Las restricciones de clave foránea deben referenciar a columnas del mismo tipo que la columna o columnas que contienen la restricción.

- Una tabla puede contener un máximo de 253 restricciones de clave foránea.

- Cada columna puede tener como máximo un único valor por defecto. No se le puede asignar un valor por defecto a una columna de tipo timestamp o con la propiedad IDENTITY.

- Una columna puede contener tantas restricciones de tipo CHECK como se quiera y cada una de estas puede tener una condición lógica tan compleja como sea necesario. Las restricciones se evaluarán en el orden en el que fueron creadas.

- Las restricciones CHECK a nivel de columna solo pueden referencias a dicha columna y las a nivel de tablas solo a columnas de dicha tabla.

- Si no se especifica si una columna admite o no nulos se siguen las siguientes reglas:
  - Si la columna está definida con un tipo definido por el usuario la columna admitirá o no nulos según admita o no nulos el tipo definido por el usuario.
  - Si la columna está definida con un tipo del sistema dependerá del tipo particular.

Ejemplos

- **Uso de restricciones de PRIMARY KEY (clave primaria)**

  job_id smallint PRIMARY KEY CLUSTERED

  emp_id smallint CONSTRAINT PK_emp_id PRIMARY KEY NONCLUSTERED

- **Uso de restricciones de FOREIGN KEY (clave ajena)**

  Job_id smallint NOT NULL DEFAULT 1 REFERENCES jobs(job_id)

  FOREIGN KEY (job_id) REFERENCES jobs(job_id)

  CONSTRAINT FK_sales_backorder FOREIGN KEY (stor_id, ord_num, title_id)
  REFERENCES sales (stor_id, ord_num, title_id)

- **Uso de restricciones UNIQUE**

  pseudonym varchar(30) NULL UNIQUE NONCLUSTERED

  CONSTRAINT U_store UNIQUE NONCLUSTERED (stor_name, city)

- **Uso de valores por defecto (DEFAULT)**

  DEFAULT 'New Position - title not formalized yet'

  DEFAULT ('9952')

DEFAULT (getdate())

DEFAULT USER

- **Uso de restricciones CHECK**

CHECK (min_lvl >= 10) and CHECK (max_lvl <= 250)

```
CONSTRAINT CK_emp_id CHECK (emp_id LIKE
  '[A-Z][A-Z][A-Z][1-9][0-9][0-9][0-9][0-9][0-9][FM]' OR
  emp_id LIKE '[A-Z]-[A-Z][1-9][0-9][0-9][0-9][0-9][FM]')

CHECK (pub_id IN ('1389', '0736', '0877', '1622', '1756')
  OR pub_id LIKE '99[0-9][0-9]')
```

- **Definición completa de tablas**

```
/****************************** jobs table ******************************/
CREATE TABLE jobs
(
  job_id          smallint          IDENTITY(1,1) PRIMARY KEY CLUSTERED,
  job_desc        varchar(50)       NOT NULL
                                    DEFAULT 'New Position - title not formalized yet',
  min_lvl         tinyint           NOT NULL CHECK (min_lvl >= 10),
  max_lvl         tinyint           NOT NULL CHECK (max_lvl <= 250)
)

/************************** employee table **************************/
CREATE TABLE employee
(
  emp_id          empid       CONSTRAINT PK_emp_id PRIMARY KEY NONCLUSTERED
                              CONSTRAINT CK_emp_id CHECK (emp_id LIKE
                              '[A-Z][A-Z][A-Z][1-9][0-9][0-9][0-9][0-9][0-9][FM]' or
                              emp_id LIKE '[A-Z]-[A-Z][1-9][0-9][0-9][0-9][0-9][FM]'),
  fname           varchar(20)         NOT NULL,
  minit           char(1)             NULL,
  lname           varchar(30)         NOT NULL,
  job_id          smallint            NOT NULL DEFAULT 1  REFERENCES jobs(job_id),
  job_lvl         tinyint             DEFAULT 10,
  pub_id          char(4)             NOT NULL DEFAULT ('9952')
                                      REFERENCES publishers(pub_id),
   hire_date      datetime            NOT NULL  DEFAULT (getdate())
)

/************************** publishers table **************************/
CREATE TABLE publishers
(
  pub_id          char(4)     NOT NULL
                              CONSTRAINT UPKCL_pubind PRIMARY KEY CLUSTERED
                              CHECK    (pub_id IN ('1389', '0736', '0877', '1622', '1756')
                                        OR pub_id LIKE '99[0-9][0-9]'),
  pub_name        varchar(40)         NULL,
  city            varchar(20)         NULL,
  state           char(2)             NULL,
  country         varchar(30)         NULL  DEFAULT('USA')
)
```

- **Uso del tipo uniqueidentifier**

```
CREATE TABLE Globally_Unique_Data
(guid            uniqueidentifier   CONSTRAINT Guid_Default  DEFAULT NEWID(),
Employee_Name    varchar(60),
CONSTRAINT Guid_PK PRIMARY KEY (Guid)
)
```

- **Uso de expresiones para crear columnas generadas**

```
CREATE TABLE mytable
  (
   low          int,
   high         int,
   myavg        AS (low + high)/2
  )
```

- **Uso de funciones del sistema para crear columnas generadas**

```
CREATE TABLE mylogintable
  (
   date_in      datetime,
   user_id      int,
   myuser_name  AS USER_NAME()
  )
```

5. MANIPULACIÓN DE TABLAS

DROP TABLE

Elimina la definición de una tabla y todos sus datos, índices, disparadores, restricciones y permisos especificados para esa tabla. Cualquier vista o procedimiento almacenado que referencia dicha tabla debe ser explícitamente borrado, la instrucciñón DROP TABLE no lo hace.

```
DROP TABLE table_name
```

Consideraciones

- DROP TABLE no borrara una tabla referenciada por una restricción de clave foránea. Primero es necesario borrar la restricción.
- No se pueden borrar tablas del sistema con la orden DROP TABLE.
- Si se borra una tabla, los elementos que lo referencian dejan de funcionar aunque una tabla con el mismo nombre sea creada.

Ejemplos

- **Borrar una tabla en la base de datos actual**

  DROP TABLE titles1

- **Borrar una tabla en otra base de datos**

  DROP TABLE pubs.dbo.authors2

ALTER TABLE

Modifica la definición de una tabla alterando, añadiendo o borrando columnas y restricciones o habilitando o deshabilitando restricciones y disparadores.

```
ALTER TABLE table
{ [ ALTER COLUMN column_name
   { new_data_type [ ( precision [ , scale ] ) ]
   [ COLLATE < collation_name > ]
   [ NULL | NOT NULL ]
   | {ADD | DROP } ROWGUIDCOL }
]
| ADD
   { [ < column_definition > ]
   | column_name AS computed_column_expression
   } [ ,...n ]
| [ WITH CHECK | WITH NOCHECK ] ADD
   { < table_constraint > } [ ,...n ]
| DROP
   { [ CONSTRAINT ] constraint_name
   | COLUMN column } [ ,...n ]
   | { CHECK | NOCHECK } CONSTRAINT
   { ALL | constraint_name [ ,...n ] }
   | { ENABLE | DISABLE } TRIGGER
   { ALL | trigger_name [ ,...n ] }
}

< column_definition > ::=
{ column_name data_type }
[ [ DEFAULT constant_expression ] [ WITH VALUES ]
| [ IDENTITY [ (seed , increment ) [ NOT FOR REPLICATION ] ] ]
]
[ ROWGUIDCOL ]
[ COLLATE < collation_name > ]
[ < column_constraint > ] [ ...n ]

< column_constraint > ::=
[ CONSTRAINT constraint_name ]
{ [ NULL | NOT NULL ]
| [ { PRIMARY KEY | UNIQUE }
[ CLUSTERED | NONCLUSTERED ]
[ WITH FILLFACTOR = fillfactor ]
[ ON { filegroup | DEFAULT } ]
]
| [ [ FOREIGN KEY ]
```

*REFERENCES ref_table [**(** ref_column **)**]*
[ON DELETE { CASCADE | NO ACTION }]
[ON UPDATE { CASCADE | NO ACTION }]
[NOT FOR REPLICATION]
]
| CHECK [NOT FOR REPLICATION]
(** logical_expression **)
}

< table_constraint > ::=
[CONSTRAINT constraint_name]
{ [{ PRIMARY KEY | UNIQUE }
[CLUSTERED | NONCLUSTERED]
*{ **(** column [**,**...n] **)** }*
[WITH FILLFACTOR = fillfactor]
[ON {filegroup | DEFAULT }]
]
| FOREIGN KEY
*[**(** column [**,**...n] **)**]*
*REFERENCES ref_table [**(** ref_column [**,**...n] **)**]*
[ON DELETE { CASCADE | NO ACTION }]
[ON UPDATE { CASCADE | NO ACTION }]
[NOT FOR REPLICATION]
| DEFAULT constant_expression
[FOR column] [WITH VALUES]
| CHECK [NOT FOR REPLICATION]
(** search_conditions **)
}

Argumentos

ALTER COLUMN
Especifica la columna que va a ser cambiada o alterada. Dicha columna no puede ser:

- Una columna con tipo **text**, **image**, **ntext**, o **timestamp**.
- Una columna definida como ROWGUIDCOL.
- Una columna generada o las columnas que la generan.
- Usada en un índice, a menos que sea del tipo **varchar**, **nvarchar**, o **varbinary**,el tipo de dato no se cambie y el nuevo tamaño sea igual o superior al antiguo.
- Usada en estadísticas.
- Usada en una restricción de clave primaria (PRIMARY KEY) o de clave ajena (FOREIGN KEY).

- Usada en una restricción CHECK o UNIQUE.

Algunos cambios pueden suponer cambios en el tipo de datos, esos cambios deben tener las siguientes consideraciones:

- El tipo de datos existente debe ser convertible al nuevo tipo de datos.
- El nuevo tipo de datos no puede ser timestamp.
- Si no se especifica nada las nuevas columnas admiten NULL.

ADD
Especifica que se añade una o más columnas o restricciones.

DROP { [CONSTRAINT] *constraint_name* | COLUMN *column_name* }
Especifica que se elimina de la tabla una columna o una restricción.

{ CHECK | NOCHECK} CONSTRAINT
Especifica que una restricción se habilita o se deshabilita.

{ENABLE | DISABLE} TRIGGER
Especifica que el disparador se activa o se desactiva.

Ejemplos

- **Añadir una nueva columna a una tabla**

```
CREATE TABLE doc_exa ( column_a INT)
GO
ALTER TABLE doc_exa ADD column_b VARCHAR(20) NULL
GO
```

- **Eliminar una columna de una tabla**

```
CREATE TABLE doc_exb ( column_a INT, column_b VARCHAR(20) NULL)
GO
ALTER TABLE doc_exb DROP COLUMN column_b
GO
```

- **Añadir una nueva columna con una restricción**

```
CREATE TABLE doc_exc ( column_a INT)
GO
ALTER TABLE doc_exc ADD column_b VARCHAR(20) NULL
  CONSTRAINT exb_unique UNIQUE
GO
```

- **Añadir una restricción** no validada **a una tabla**

```
CREATE TABLE doc_exd ( column_a INT)
GO
INSERT INTO doc_exd VALUES (-1)
```

```
GO
ALTER TABLE doc_exd WITH NOCHECK
ADD CONSTRAINT exd_check CHECK (column_a > 1)
GO
```

- **Añadir varias columnas con restricciones**

```
CREATE TABLE doc_exe ( column_a INT CONSTRAINT column_a_un UNIQUE)
GO
ALTER TABLE doc_exe ADD
/* Añadir una columna como clave primaria */
column_b INT IDENTITY CONSTRAINT column_b_pk PRIMARY KEY,

/* Añadir una columna como clave foránea de la misma tabla. */
column_c INT NULL  CONSTRAINT column_c_fk  REFERENCES doc_exe(column_a),

/* Añadir una columna con una restricción de comprobacion */
column_d VARCHAR(16) NULL  CONSTRAINT column_d_chk
CHECK (column_d IS NULL OR
        column_d LIKE "[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]" OR
        column_d LIKE "([0-9][0-9][0-9]) [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]"),

/* Añadir una columna con valor por defecto  */
column_e DECIMAL(3,3) CONSTRAINT column_e_default DEFAULT .081
GO
```

- **Añadir una columna con valor por defecto y NULL**

```
ALTER TABLE MyTable
ADD AddDate smalldatetime NULL
CONSTRAINT AddDateDflt DEFAULT getdate() WITH VALUES
```

- **Deshabilitar y habilitar una restricción**

```
CREATE TABLE cnst_example
(id INT NOT NULL,
 name VARCHAR(10) NOT NULL,
 salary MONEY NOT NULL
    CONSTRAINT salary_cap CHECK (salary < 100000)
)

-- Inserciones validas
INSERT INTO cnst_example VALUES (1,"Joe Brown",65000)
INSERT INTO cnst_example VALUES (2,"Mary Smith",75000)

-- Inserción que viola la restricción
INSERT INTO cnst_example VALUES (3,"Pat Jones",105000)

-- Deshabilitar la restricción
ALTER TABLE cnst_example NOCHECK CONSTRAINT salary_cap
INSERT INTO cnst_example VALUES (3,"Pat Jones",105000)

-- Habilitar la restricción
ALTER TABLE cnst_example CHECK CONSTRAINT salary_cap
INSERT INTO cnst_example VALUES (4,"Eric James",110000)
```

INSTRUCCIÓN SELECT

Aunque la instrucción SELECT se utiliza principalmente para recuperar datos específicos, también se puede utilizar para asignar un valor a una variable local o para llamar a una función, como se verá en la sección «Otros usos de SELECT». Una instrucción SELECT puede ser simple o compleja (no es necesariamente mejor que sea compleja). Hay que intentar construir las instrucciones SELECT de la forma más sencilla posible siempre que recuperen los resultados necesarios. Por ejemplo, si se necesitan datos de solamente dos columnas de una tabla, hay que incluir solamente esas dos columnas en la instrucción SELECT para minimizar la cantidad de datos que se deben devolver.

Después de que se haya decidido qué datos se necesitan y de qué tablas, se puede determinar qué otras opciones, en caso necesario, hay que utilizar. Estas opciones pueden incluir especificar qué columnas deberían estar en la cláusula WHERE para utilizar los índices, especificar si hay que ordenar los datos devueltos y especificar si se han de devolver solamente valores distintos.

Comencemos examinando las diversas opciones para la instrucción SELECT y revisemos ejemplos de cada una. Las bases de datos ejemplo utilizadas en este capítulo, pubs y Northwind, se crearon de forma automática cuando se instaló Microsoft SQL Server 2000. Para familiarizarse con las bases de datos pubs y Northwind se puede utilizar el Administrador corporativo SQL Server para examinar las tablas de las bases de datos.

Cuando se ejecuta la instrucción SELECT interactivamente (por ejemplo, utilizando OSQL o el Analizador de consultas SQL los resultados se muestran en columnas, con una cabecera de columna por claridad.

La sintaxis para la instrucción SELECT consiste en varias cláusulas, la mayoría de las cuales son opcionales. Una instrucción SELECT debe incluir al menos una cláusula SELECT y una cláusula FROM. Las dos cláusulas identifican qué columna o columnas de datos recuperar y desde qué tabla o tablas recuperar los datos, respectivamente. Las instrucción SELECT consta de las siguientes cláusulas, las cuales deben ir en el orden que se ve a continuación:

SELECT
INTO
FROM
WHERE
GROUP BY
HAVING
ORDER BY

CLÁUSULA SELECT

La cláusula SELECT consiste en una lista de selección y posiblemente algunos argumentos opcionales. La *lista de selección* es la lista de expresiones o columnas que se especifican en la cláusula SELECT para indicar qué datos hay que devolver. Se describen en esta sección los argumentos opcionales y la lista de selección.

Argumentos

Se pueden utilizar los siguientes dos argumentos en la instrucción SELECT para controlar que filas se devuelven:

- **DISTINCT** Devuelve solamente filas únicas. Si la lista de selección contiene varias columnas, las filas se considerarán únicas si los valores correspondientes en al menos una de las columnas son diferentes. Para que se dupliquen dos filas deben contener valores idénticos en cada columna.

- **TOP n [PERCENT]** Devuelve solamente las primeras filas del conjunto resultado. Si se especifica PERCENT, solamente el primer porcentaje de las filas se devuelve. Cuando se utiliza PERCENT, debe estar entre 0 y 100. Si la consulta incluye una cláusula ORDER BY, las filas se ordenan primero y posteriormente se devuelve el porcentaje del conjunto resultado ordenado.

El siguiente código T-SQL muestra un ejemplo de la instrucción SELECT que se ejecuta tres veces, cada vez con un argumento diferente. La primera consulta utiliza el argumento DISTINCT, la segunda columna utiliza el argumento TOP 50 PERCENT y la tercera columna utiliza el argumento TOP 5.

```
SELECT    DISTINCT au_fname,au_lname
FROM      authors
GO
SELECT    TOP 50 PERCENT au fname,au_lname
FROM      authors
GO
SELECT    TOP 5 au_fname,au_lname
FROM      authors
GO
```

La primera consulta devuelve 23 filas, cada una de las cuales es única. La segunda consulta devuelve 12 filas (aproximadamente el 50 por ciento, redondeado) y la tercera consulta devuelve 5 fias.

La lista de selección

Como se ha mencionado, la lista de selección (select list) es la lista de expresiones o columnas que se especifica en la cláusula SELECT para indicar qué datos se han de devolver. La expresión pueden ser una lista de nombres de columnas, funciones o constantes. La lista de selección puede incluir varias expresiones o nombres de columnas, separados por comas. El ejemplo anterior utiliza la lista de selección.

```
au_fname,au_lname
```

Signo *, o carácter comodín: Se puede utilizar el asterisco (*) o carácter comodín en la lista de selección para devolver todas las columnas de todas las tablas y vistas llamadas en la cláusula FROM. Por ejemplo, para devolver todas las columnas de todas las filas de la tabla *sales* en la base de datos *pubs* se puede utilizar la siguiente consulta:

```
SELECT      *
FROM        sales
GO
```

En la sección «Combinaciones cruzadas» se describe qué sucede cuando se lista más de una tabla en la cláusula FROM de una instrucción SELECT que contiene el carácter comodín.

IDENTITYCOL y ROWGUIDCOL: Para recuperar los valores de una columna identidad de una tabla se puede simplemente utilizar la expresión IDENTITYCOL en la lista de selección.

En el siguiente ejemplo se consulta la base de datos Northwind, la cual tiene una columna identidad definida en la tabla *Employees:*

```
USE    Northwind
GO
SELECT      IDENTITYCOL
FROM        Employees
GO
```

El resultado será el siguiente:

```
EmployeeID
---------------
3
4
8
.
.
.
9
(9 filas afectadas)
```

Nótese que la cabecera de la columna en el resultado coincide con el nombre de la columna en la tabla que tiene la propiedad IDENTITY (en este caso, *EmployeeID).* De una forma similar se puede utilizar la expresión ROWGUIDCOL en la lista de selección para recuperar valores de la columna del identificador único global (GUID, Globally Unique Identifier) (esto es, la columna que tiene la propiedad ROWGUIDCOL). Para que una columna tenga la propiedad ROWGUIDCOL debe ser del tipo de datos *uniqueidentifier*.

Alias de columna: El uso de una columna alias en la lista de selección permite especificar la cabecera de la columna que se desea que aparezca en el resultado. Se puede utilizar un alias para clarificar el significado de los datos en una columna de salida, para asignar una cabecera a una columna que se utiliza en una función y para referirse a una cláusula ORDER BY**.**

Cuando existen dos o más columnas con el mismo nombre en tablas distintas se puede querer incluir el nombre de la tabla en la cabecera de la columna de salida para una mayor claridad. Como ejemplo de utilización de una columna alias se verá la columna *lname* en la tabla *employee* de la base de datos *pubs*. Se puede realizar la siguiente consulta:

```
USE    pubs
GO
SELECT      lname
FROM        employee
GO
```

Si se hace dicha consulta se obtendrán los siguientes resultados:

```
lname
-------------
Cruz
Roulet
Devon
....
O' Rourke
Ashworth
Latimer
(43 filas las afectadas)
```

Para mostrar la cabecera «Apellido del empleado» en lugar de la cabecera original *lname* en el resultado (para enfatizar el hecho de que el apellido viene de la tabla *employee)* se utiliza la palabra clave AS, como se muestra seguidamente:

```
SELECT      lname AS «Employee Last Name»
FROM        employee
GO
```

El resultado de esta consulta es:

```
Employee Last Name
------------------------------
Cruz
Roulet
Devon
...
O'Rourke
Ashworth
Latimer
(43 filas afectadas)
```

También se puede utilizar un alias de columna con otros tipos de expresiones en la lista de selección y como una columna de referencia en una cláusula ORDER BY Supóngase que se hace una llamada a una función en la lista de selección. Para asignar un alias de columna que describe la salida de la función hay que utilizar la palabra clave AS después de la llamada a la función. Si no se utiliza un alias con una función, no habrá ninguna cabecera de columna. Por ejemplo, la siguiente instrucción asigna la cabecera de columna «Maximum Job ID» para la salida de la función MAX:

```
SELECT    MAX(job_id)AS «Maximum Job ID»
FROM      employee
GO
```

El alias de columna se encierra entre comillas puesto que contiene varias palabras con espacios entre ellas. Si el alias no tiene espacios no se necesitan las comillas, como se verá en el siguiente ejemplo.

Se puede referir a un alias de columna que se asignó en la cláusula SELECT como un argumento en la cláusula ORDER BY. Esta técnica es útil cuando la lista de selección contiene una función cuyos resultados se tienen que ordenar. Por ejemplo, la siguiente instrucción recupera la cantidad de libros vendidos en cada tienda y ordena la salida según la cantidad. El alias asignado en la lista de selección se utiliza en la cláusula ORDER BY:

```
SELECT    SUM(qty) AS Quantity_of_Books,stor_id
FROM      sales
GROUP BY  stor_id
ORDER BY  Quantity_of_Books
GO
```

En este caso, no se ha encerrado el alias entre comillas puesto que no contiene espacios. Si no hubiese asignado un alias de columna para *SUM(qty)* en esta consulta, podríamos haber utilizado *SUM(qty)* en lugar del alias en la cláusula ORDER BY Esta técnica, mostrada en el siguiente ejemplo, proporcionará la misma salida, pero sin cabecera de columna para columna SUM:

```
SELECT    SUM(qty) stor_id
FROM sales
GROUP BY  stor_id
ORDER BY  SUM(qty)
GO
```

Hay que recordar que se utiliza un alias de columna para asignar una cabecera a una columna por claridad de la salida; el resultado de la consulta no afecta en manera alguna.

CLÁUSULA FROM

La cláusula FROM contiene los nombres de las tablas y vistas de las cuales se obtienen los datos. Cada instrucción SELECT requiere una cláusula FROM, excepto cuando la lista de selección no contiene nombres de columna (solamente constantes, variables y expresiones aritméticas). También se han visto algunos ejemplos sencillos con la cláusula FROM, pero las cláusulas FROM también pueden contener tablas derivadas, combinaciones y alias.

Tablas derivadas

Una *tabla derivada* es el resultado de una instrucción SELECT anidada en la cláusula FROM. El resultado de la instrucción SELECT anidada se utiliza como una tabla desde la cual la instrucción SELECT exterior selecciona los datos. La siguiente consulta utiliza una tabla derivada para encontrar los nombres de cualquier tienda que realiza al menos un tipo de descuento:

```
USE    pubs
GO
SELECT      s.stor_name
FROM        stores AS s, (SELECT stor_id,COUNT(DISTINCT discounttype)
                          AS d_count
                          FROM discounts
                          GROUP BY stor_id) AS d
WHERE       s.stor_id =d.stor_id
AND         d.d_count >=1
GO
```

Si se ejecutan estas instrucciones se debería mostrar una fila seleccionada, la cual significa que solamente una tienda en la base de datos, Bookbeat, ofrece descuento.

Nótese que esta consulta utiliza abreviaturas para los nombres de las tablas (s para la tabla *stores* y d para la tabla *discounts).* Esta abreviatura, denominada un *alias de tabla* se describe en la sección «Alias de tablas» en este capítulo.

Nota: No se puede tener una tabla derivada en una cláusula WHERE.

Tablas combinadas

Una *tabla combinada* es el resultado de una operación de combinación realizada sobre dos o más tablas. Se pueden realizar varios tipos de combinaciones de tablas: combinaciones internas (inner), externa completa (full outer), externa por la izquierda (left outer), externa por la derecha (right outer) y cruzada (cross). Veamos detalladamente cada una de estas combinaciones.

Combinaciones internas: Una *combinación interna (Inner join)* es el tipo de combinación predeterminado; especifica que solamente se han de incluir en el resultado filas de la tabla que satisface la condición ON. Para especificar una combinación hay que utilizar la palabra clave JOIN. Se utiliza la palabra clave ON para identificar la

condición de búsqueda sobre la cual se basa la combinación. La siguiente consulta combina las tablas *stores y discounts* para mostrar qué tienda ofrece un descuento y el tipo de descuento (de forma predeterminada, esto es una combinación interna, lo que significa que solamente se devuelven filas que satisfacen la condición de búsqueda ON).

```
SELECT      s.stor_id,d.discounttype
FROM        stores s JOIN discounts d
ON          s.stor_id =d.stor_id
GO
```

El resultado puede ser el siguiente:

```
stor_id     discounttype
---------   ----------------
8042        Customer Discount
```

Como se puede ver, solamente una tienda ofrece un descuento y tiene únicamente un tipo de descuento. La única fila devuelta es aquella cuyo *stor_id* de la tabla *stores* tiene un *stor_id* que coincida de la tabla *discounts.* Se devuelve el *stor_id* particular y su *discounttype* asociado.

Combinaciones externas completas: Una *combinación externa completa (full outer join)*especifica que se deberían incluir en el resultado las filas no coincidentes (filas que no cumplen la condición ON) así como las filas que coincidan (filas que cumplen la condición ON). Para filas que no coincidan, aparecerá *NULL* en la columna de no coincidencia. En este ejemplo, *NULL* significa tanto que una tienda no ofrece ningún descuento, y por ello tiene un valor *stor_id* en la tabla *stores* pero no en la tabla *discounts*, como que el tipo de descuento en la tabla *discounts* no se ofrece en ninguna tienda. La siguiente consulta utiliza la misma consulta que en la combinación interna anterior, pero esta vez especificará FULL OUTER JOIN:

```
SELECT      s.stor_id,d.discounttype
FROM        stores s FULL OUTER JOIN discounts d
ON          s.stor_id =d.stor_id
GO
```

El resultado será:

```
stor_id          discounttype
----------       ----------------
NULL             Initial Customer
NULL             Volume Discount
6380             NULL
7066             NULL
7067             NULL
7131             NULL
7896             NULL
8042             Customer Discount
```

Solamente una de las filas de los resultados muestra una coincidencia (la última fila). Las otras filas tienen NULL en una columna.

Combinaciones externas por la izquierda: Una *combinación externa por la izquierda (left outer join)* devuelve las filas coincidentes más todas las filas de la tabla

que se especifican a la izquierda de la palabra clave JOIN. Utilizando la misma consulta, especificamos esta vez LEFT OUTER JOIN, como se muestra aquí:

```
SELECT    s.stor_id,d.discounttype
FROM      stores s LEFT OUTER JOIN discounts d
ON        s.stor_id =d.stor_id
GO
```

El resultado será:

```
stor_id          discounttype
---------        ----------------
6380             NULL,
7066             NULL,
7067             NULL
7131             NULL
7896             NULL
8042             Customer Discount
```

Este resultado incluye las filas de la tabla *stores* que no tiene un valor *stor_id* coincidente en la tabla *discounts* (La columna *discounttype* para estas filas es *NULL).* El resultado también incluye la fila que satisface la condición ON.

Combinaciones externas por la derecha: Una *combinación externa por la derecha (right outer join)* es lo contrario a una combinación externa por la izquierda: devuelve las filas coincidentes más todas las filas de la tabla especificada a la derecha de la palabra clave JOIN. Veamos la misma consulta con RIGHT OUTER JOIN:

```
SELECT    s.stor_id,d.discounttype
FROM      stores s RIGHT OUTER JOIN discounts d
ON        s.stor_id =d.stor_id
GO
```

El resultado es:

```
stor_id          discounttype
-----------      ----------------------
NULL             Initial Customer
NULL             Volume Discount
8042             Customer Discount
```

Este resultado muestra las filas de la tabla *discounts* que no tienen un valor *stor_id* coincidente en la tabla *stores* (la columna *stor_id* para esas filas es *NULL).* El resultado también muestra una fila que satisface la condición ON.

Combinaciones cruzadas: Una *combinación cruzada (cross join) es* el producto de dos tablas cuando no se especifica la cláusula WHERE. Cuando se especifica una cláusula WHERE la combinación cruzada actúa como una combinación interna. Sin una cláusula WHERE se devolverán de ambas tablas todas las filas y columnas de la siguiente forma: cada fila de la primera tabla se unirá con cada fila de la segunda tabla, de forma que el tamaño del resultado será el número de filas de la primera tabla multiplicado por el número de filas de la segunda tabla.

Para entender una combinación cruzada, veremos algunos ejemplos nuevos. Primeramente veremos una combinación cruzada sin una cláusula WHERE y posteriormente veremos tres ejemplos de combinaciones cruzadas que incluyen cláusulas WHERE. Las siguientes consultas muestran un ejemplo simple. Al ejecutar las tres consultas hay que fijarse en el número de filas de cada resultado.

```
SELECT    *
FROM      stores
GO
SELECT    *
FROM      sales
GO
SELECT    *
FROM      stores CROSS JOIN sales
GO
```

Si se incluyen dos tablas en la cláusula FROM el efecto es el mismo que al especificar CROSS JOIN, como en el siguiente ejemplo:

```
SELECT    *
FROM      stores,sales
GO
```

Para evitar esta cantidad de información (si es más de lo que necesitarnos) podemos agregar una cláusula WHERE para estrechar la consulta, como en la siguiente instrucción:

```
SELECT    *
FROM      sales CROSS JOIN stores
WHERE     sales.stor_id =stores.stor_id
GO
```

Esta instrucción devuelve solamente las filas que satisfacen la condición de busqueda en la cláusula WHERE, lo cual estrecha el resultado a 21 filas. La cláusula WHERE fuerza a una combinación cruzada a actuar de la misma forma que una combinación interna (es decir, se devuelven solamente filas que satisfacen la condición de búsqueda). La consulta anterior devuelve las filas de la tabla *sales,* concatenadas con las filas de la tabla *stores* que tienen el mismo valor *stor_id* Las filas no coincidentes no se devuelven.

Para reducir más el resultado, se puede especificar de qué tabla seleccionar todas las filas y columnas añadiendo el nombre de la tabla antes del asterisco (*), como se muestra en la siguiente consulta. También se puede especificar a qué tabla pertenece una columna insertando el nombre de la tabla y un punto (.) antes de cualquier nombre de la columna.

```
SELECT    sales.*,stores.city
FROM      sales CROSS JOIN stores
WHERE     sales.stor_id =stores.stor_id
GO
```

Esta consulta devuelve todas las columnas de la tabla *sales*, con la columna *city* de la fila de la tabla *stores* que tiene el mismo valor *stor_id* anexionado. En efecto, el resultado incluye la ciudad de la tienda donde se realizó la venta anexionado a las filas de la tabla *sales* que tienen un valor *stor_id* coincidente en la tabla *stores.*

Aquí se muestra la misma consulta sin el símbolo solamente se seleccionará la columna *stor_id* de la tabla *sales:*

```
SELECT      sales.stor_id,stores.city
FROMsales CROSS JOIN stores
WHERE       sales.stor_id =stores.stor_id
GO
```

Alias de tablas

Ya se han visto varios ejemplos en los cuales se utiliza un alias de nombre de tabla. Especificar palabra clave AS es opcional (FROM *nombretabla* AS *alias* produce el mismo resultado que FROM *nombretabla alias).* Veamos de nuevo la consulta de la sección «Combinaciones externas por la derecha», que utilizaba alias:

```
SELECT      s.stor_id,d.discounttype
FROM        stores s RIGHT OUTER JOIN discounts d
ON          s.stor_id =d.stor_id
GO
```

Cada una de las dos tablas tiene una columna *stor_id* Para distinguir qué columna de la tabla *stor_id* se está refiriendo en la consulta se debe proporcionar el nombre de la tabla o un alias seguido de un punto (.) y después el nombre de la columna. En este ejemplo se utiliza el alias *s* para la tabla *stores y d* para la tabla *discounts.* Cuando se especifica una columna, hay que agregar una s. o d. antes del nombre de la columna para indicar qué tabla la contiene. Veamos la misma consulta con la palabra clave AS:

```
SELECT      s.stor_id,d.discounttype
FROM        stores AS s RIGHT OUTER JOIN discounts AS d
ON          s.stor_id =d.stor_id
GO.
```

CLÁUSULA INTO

Ésta es realmente la primera cláusula realmente opcional para la instrucción SELECT: la cláusula INTO. Al usar la sintaxis SELECT <*lista_de_selección*> INTO <*nueva_tabla*> se permite recuperar datos de una tabla o tablas y ubicar las filas resultado en una nueva tabla. La nueva tabla se crea automáticamente cuando se ejecuta la instrucción SELECT ... INTO y se defina según las columnas en la lista de selección. Cada columna en la nueva tabla tiene el mismo tipo de datos que la columna original y lleva el nombre de la columna especificado en la lista de selección. El usuario debe

tener el permiso CREATE TABLE en la base de datos de destino para ejecutar SELECT ... INTO.

Se puede utilizar SELECT ... INTO para seleccionar filas en una tabla temporal o en una tabla permanente. Para una tabla local temporal (que es visible solamente a la conexión actual o usuario) se debe incluir el símbolo (#) antes del nombre de la tabla. Para una tabla temporal global (que es visible a cualquier usuario) se deben incluir dos símbolos (##) antes del nombre de la tabla. Una tabla temporal se borra automáticamente después de que todos los usuarios que están utilizando la tabla se hayan desconectado de SQL Server. Para seleccionar en una tabla permanente no se necesita un prefijo para el nuevo nombre de la tabla, pero se debe activar la opción Select Into/Bulk Copy en la base de datos de destino. Para activar esta opción en la base de datos pubs, se puede ejecutar la siguiente instrucción OSQL:

```
sp_dboption pubs,"select into/bulkcopy",true
GO
```

También se puede utilizar el Administrador corporativo SQL Server para activar esta opción, según se sigue:

1. Pulsar el botón derecho del ratón sobre el nombre de la base de datos pubs en el panel del Administrador corporativo y elegir Propiedades del menú contextual para mostrar la ventana Propiedades de Pubs.

2. Pulsar la ficha Opciones, según se muestra en la figura siguiente y seleccionar Registro masivo en la lista desplegable Modelo. Dejar el resto de otros parámetros como están. Pulsar Aceptar.

La siguiente consulta utiliza SELECT ... INTO para crear una tabla permanente, *emp_info*, que incluye los nombres y apellidos de todos los empleados y la descripción de su puesto de trabajo (de la base de datos pubs):

```
SELECT    employee.fname,employee.lname,jobs.job_desc
INTO      emp_info
FROM      employee,jobs
WHERE     employee.job_id =jobs.job_id
GO
```

La tabla *emp_info* contendrá tres columnas *(fname, lname y job_desc)* que tiene los mismos tipos de datos que en las columnas definidas en las tablas originales *(employee y jobs).* Si se desea que la nueva tabla sea una tabla temporal, la tabla debe tener un símbolo # precedido, corno en *#emp_info;* para una tabla temporal global, hay que utilizar el símbolo ## como en *##emp_info.*

CLÁUSULA WHERE Y CONDICIONES DE BÚSQUEDA

Se puede utilizar la cláusula WHERE para restringir las filas que se devuelven de una consulta según las condiciones de búsqueda especificadas. En esta sección, examinaremos muchas operaciones que se pueden utilizar en la condición de búsqueda.

En primer lugar revisaremos algo de terminología. La condición de búsqueda puede contener un número ilimitado de predicados combinados mediante los operadores lógicos *AND, OR y NOT*. Un *predicado* es una expresión que devuelve un valor *TRUE, FALSE o UNKNOWN*. Una *expresión* puede ser un nombre de columna, constante, función escalar (una función que devuelve un valor), variable, subconsulta escalar (una subconsulta que devuelve una columna) o una combinación de estos elementos combinados por operadores. En esta sección el término «expresión» se refiere a predicados y expresiones.

Operadores de comparación

En la tabla siguiente se muestran los operadores de igualdad y desigualdad que se pueden utilizar.

Operación	Condición comprobada
=	Analiza la igualdad entre dos expresiones
<>	Analiza si dos expresiones no son iguales
!=	Analiza si dos expresiones no son iguales (igual que <>)
>	Analiza si una expresión es mayor que otra
>=	Analiza si una expresión es mayor o igual que otra
!>	Analiza si una expresión no es mayor que otra
<	Analiza si una expresión es menor que otra
<=	Analiza si una expresión es menor o igual que otra
!<	Analiza si una expresión no es menor que otra

Una cláusula sencilla WHERE puede comparar dos expresiones utilizando el operador igualdad (=). Por ejemplo, la siguiente instrucción SELECT analiza el valor en la columna *lname* para cada fila, que es del tipo *char* y devuelve *TRUE* si el valor es igual a *«Latimer»* (las filas que devuelven *TRUE* se incluirán en el resultado).

```
SELECT    *
FROM      employee
WHERE     lname ="Latimer"
```

En este caso, la consulta devuelve una fila. El nombre Latimer debe estar entre paréntesis, puesto que es una cadena de caracteres.

Nota: *De forma predeterminada, SQL Server aceptará tanto comillas simples(‘ ’) como comilla dobles (“”) que establecer la opción a TRUE (FALSE es la forma predeterminada).*

La siguiente consulta utiliza el operador de desigualdad (<>), esta vez con una columna, job_id del tipo de datos *entero:*

```
SELECT    job_desc
FROM      jobs
WHERE     job_id <>1
GO
```

Esta consulta devolverá el texto de descripción del puesto de trabajo de la fila o filas de la tabla jobs que tiene un valor *job_id* distinto a 1. En este caso se devuelven 13 filas. Si una fila tiene un valor de *NULL* no es igual a ni a ningún otro valor, de forma que las filas con valores nulos también se devolverán.

Operadores lógicos

Los operadores lógicos AND y OR verifican dos expresiones y devuelven el valor booleano *TRUE, FALSE o UNKNOWN* según los resultados de las dos expresiones. El operador NOT niega el valor booleano devuelto por la expresión que lo sigue. La figura anterior muestra el valor devuelto por cada operación AND, OR y NOT posible. Para leer las tablas AND y OR, hay que buscar el resultado de la primera expresión en la columna izquierda, buscar el resultado de la segunda expresión en la fila superior y entonces buscar la celda en la cual la fila y columna se encuentran para ver el valor booleano resultante. La tabla NOT es bastante sencilla. Puede resultar un valor *UNKNOWN* de una expresión que contenga *NULL* como operando.

La siguiente consulta utiliza dos expresiones en la cláusula WHERE con el operador lógico AND:

```
SELECT    job_desc,min_lvl,max_lvl
FROM      jobs
WHERE     min lvl >=100 AND
          max_lvl <=225
GO
```

Como se muestra en la figura anterior, para que una operación AND devuelva *TRUE,* ambas condiciones deben devolver *TRUE.* En esta consulta se devuelven cuatro filas.

En la siguiente consulta, una operación OR verifica los editores en Washington D.C. o en Massachusetts. Se devolverá una fila si una de las verificaciones devuelve *TRUE* para cada fila.

```
SELECT    p.pub_name,p.state,t.titie
FROM      publishers p,titles t
WHERE     p.state ="DC" OR
          p.state ="MA" AND
          t.pub_id =p.pub_id
GO
```

Esta consulta devuelve 23 filas.

La operación NOT simplemente devuelve la negación del valor de la expresión booleana que la sigue. Por ejemplo, para devolver todos los títulos de los libros de aquellos autores cuyos derechos de autor son menores a un 20 por ciento, se puede utilizar el operador NOT de la siguiente manera:

```
SELECT    t.title,r.royalty
FROM      titles t,roysched r
WHERE     t.title_id =r.title_id AND
          NOT r.royalty <20
GO
```

Esta consulta devuelve los 18 títulos para los cuales los derechos de autor sean iguales o mayores del 20 por ciento.

Otras palabras claves

Además de los operadores descritos en secciones anteriores, se pueden utilizar una serie de palabras claves T-SQL en una condición de búsqueda. Se explican en esta sección las palabras claves más utilizadas y se dan ejemplos de su uso.

LlKE: La palabra clave LlKE indica el patrón de ajuste con una condición de búsqueda. La *coincidencia de patrones* significa analizar una coincidencia entre una expresión y el patrón especificado en la condición de búsqueda, según la siguiente sintaxis:

<expresión> LIKE <patrón>

Si la expresión coincide con el patrón se devuelve un valor booleano *TRUE*. En caso contrario se devuelve *FALSE*. La expresión debe ser del tipo de datos *cadena de caracteres.* Si no lo es, SQL Server la convertirá al tipo de datos *cadena de caracteres*, si es posible.

Los patrones son realmente expresiones de cadena. Se define una *expresión de cadena* como una cadena de caracteres que pueden incluir comodines. Los *caracteres comodín* son caracteres que adoptan significados especiales cuando se utilizan en una expresión de cadena. La tabla siguiente lista los caracteres comodín que se pueden utilizar en patrones.

Carácter comodín	Descripción
%	Símbolo de porcentaje; coincide con una cadena de cero o más caracteres.
_	Subrayado; coincide con un único carácter.
[]	Carácter comodín en rango; coincide con cualquier carácter único en el rango o conjunto, tales como [m-p] o [mnop], significando que cualquiera de los caracteres puede ser m, n, o, p.
[^]	Carácter comodín no en rango; coincide con cualquier carácter distinto de m, n, o, p.

Para tener un mejor conocimiento del uso de la palabra clave LlKE y de los caracteres comodín se verán algunos ejemplos. Para encontrar todos los apellidos en la tabla *authors* que conmienzan con la letra «S» se puede utilizar la siguiente consulta con el carácter comodín %:

```
SELECT      au_lname
FROM        authors
WHERE       au_lname LlKE "S%"
GO
```

El resultado será:

```
au_lname
---------------
Smith
Straight
Stringer.
```

En esta consulta «S%» significa que se devuelven todas las filas que contienen un apellido que comienza por una «S» seguido por cualquier número de caracteres.

Nota: Los ejemplos en esta sección asumen que se está utilizando el orden del diccionario sin importar mayúsculas. Si se ha especificado otro orden, los resultados pueden ser diferentes, pero la teoría de operación de la palabra clave LlKE seguirá siendo la misma.

Para recuperar la información de un autor cuyo ID comienza por el número 724, sabiendo que cada ID tiene el formato de tres dígitos seguido por un guión, seguido por dos dígitos, otro guión y finalmente cuatro dígitos se puede utilizar el carácter comodín _ según sigue:

```
SELECT      *
FROM        authors
WHERE       au_id LIKE "724-___-____”
GO
```

El resultado contendrá dos filas con los valores *au_id* de *742-08-9931* y *724-80-9391*.

Ahora veremos un ejemplo que utiliza el comodín []. Para recuperar los apellidos de los autores cuya primera letra comienza entre «A» y «M» se puede utilizar el comodín [] junto con el carácter comodín % como se muestra aquí:

```
SELECT      au_lname
FROM        authors
WHERE       au_lname LIKE "[A-M] %”
GO
```

El resultado contendrá 14 filas de nombres que comienzan entre «A» y «M» (13, si se está utilizando un orden que distinga mayúsculas).

Si se realiza una consulta similar pero se usa el carácter comodín [^] en lugar del carácter comodín [] se obtendrán filas que contienen apellidos que comienzan con letras distintas a «A» entre «M», como se muestra seguidamente:

```
SELECT    au_lname
FROM      authors
WHERE     au_lname LIKE "[^A-M ] %"
GO
```

Esta consulta devuelve nueve filas.

Si se está utilizando un orden que distinga mayúsculas y se desean encontrar todos los nombres que caen en un rango sin importar la caja, se puede utilizar una consulta que comprueba si la primera letra es minúscula o mayúscula, como se muestra aquí:

```
SELECT    au_lname
FROM      authors
WHERE     au_lname LIKE " [A-M ] %" OR
          au_lname LIKE " [a-m ] %"
GO
```

Este resultado incluirá el nombre «del Castillo», mientras que en una consulta que distinga la caja que busca solamente mayúsculas entre «A» y «M» no.

La palabra clave LlKE también puede venir precedida por el operador NOT. NOT LlKE devuelve filas que no coinciden con la condición especificada. Por ejemplo, para seleccionar títulos que no comiencen por la palabra «The», se puede utilizar NOT LlKE en la siguiente consulta:

```
SELECT    title
FROM      titles
WHERE     title NOT LIKE "The %”
GO
```

Esta consulta devuelve 15 filas.

Se puede ser creativo en el uso de la palabra clave LlKE. Sin embargo hay que tener la precaución de verificar que las consultas devuelven los datos que se esperan. Si no se escribe un NOT o un carácter ^ cuando se tiene intención de hacerlo el resultado puede ser el opuesto al deseado. No incluir el carácter comodín % cuando es necesario producirá también resultados incorrectos. Hay que recordar que los espacios al principio y al final también deben coincidir exactamente.

ESCAPE: La palabra clave ESCAPE permite realizar coincidencia de patrones con los propios caracteres comodín tales como, %, y _. A continuación de la palabra clave ESCAPE se especifica el carácter que se desee usar como carácter de escape, que significa que el siguiente carácter en la expresión de cadena debería coincidir literalmente. Por ejemplo, para buscar todas las filas en la tabla titles que tienen un subrayado en la columna *title* se puede realizar la siguiente consulta:

```
SELECT    title
FROM      titles
WHERE     title LIKE "%e_%"  'ESCAPE “e”
GO
```

Esta consulta no devuelve filas, puesto que no hay ningún título en la base de datos que incluya un subrayado.

BETWEEN: La palabra clave BETWEEN se utiliza siempre con AND y especifica un rango inclusivo para verificar una condición de búsqueda. La sintaxis es la siguiente:

<expresión de comprobación>BETWEEN <expresión inicial>AND <expresión final>

El resultado de una condición de búsqueda será el valor booleano *TRUE si expresión de comprobación* es mayor o igual a expresión inicial y menor o igual que *expresión final.* En otro caso el resultado será *FALSE.*

La siguiente consulta utiliza BETWEEN para encontrar todos los títulos de libros que tienen un precio entre 5 y 25.

```
SELECT    price,title
FROM      titles
WHERE     price BETWEEN 5.00 AND 25.00
GO
```

Esta consulta devuelve 14 filas.

También se puede utilizar NOT con BETWEEN para buscar filas que no están en el rango especificado. Por ejemplo, para encontrar los títulos de libros cuyos precios no están entre 20 y 30 (es decir que su precios son menores a 20 o mayores a 30) se puede utilizar la siguiente consulta:

```
SELECT    price,title
FROM      titles
WHERE     price NOT BETWEEN 20.00 AND 30.00
GO
```

Cuando se utilizan la palabra clave BETWEEN, *expresión de comprobación* debe tener mismo tipo de datos que *expresión inicial y expresión final.*

En el *ejemplo anterior*, la columna *price* tiene el tipo de datos *moneda*, de forma que *expresión inicial y expresión final* deben ser un número que se puede comparar o convertir de forma implícita al tipo de datos *moneda.* No se puede utilizar *price* como *expresión de comprobación* y después utilizar una cadena de caracteres (del tipo *char)* con *expresión inicial y expresión final.* Si se hace, SQL Server devolverá un mensaje de error.

Nota: *SQL Server convertirá automáticamente los tipos de datos cuando sea necesario sí es posible una conversión implícita. La conversión implícita es la conversión automática de un*

tipo de datos a otro, siempre que sean tipos de datos compatibles. Después de la conversión se puede realizar la comparación. Por ejemplo, si se compara una columna con el tipo de datos smallint *con una columna del tipo de datos* int, *SQL Server implícitamente convierte el tipo de datos de la primera columna a* int *antes de realizar la comparación. Si no se soporta una conversión implícita, se puede utilizar la función CAST o CONVERT para convertir explícitamente una columna. Una tabla completa con los tipos de datos que SQL Server convertirá implícitamente y con los que son necesario convertir explícitamente se puede ver «CAST» en el índice de Libros en pantalla de SQL Server (SQL Server Books Online) y seleccionar <CAST y CONVERT (T-SQL),, en el cuadro de diálogo Temas encontrados.*

Nuestro último ejemplo que involucra a la palabra clave BETWEEN utiliza cadenas en una condición de búsqueda. Para encontrar los apellidos de los autores que están entre los nombres «Bennet» y «McBadden» se puede utilizar la siguiente consulta:

```
SELECT    au lname
FROM      authors
WHERE     au_lname BETWEEN "Bennet" AND "McBadden"
GO
```

Puesto que el rango BETWEEN es inclusivo, el resultado de esta consulta incluirá los nombres «Bennet» y «McBadden», los cuales existen en la tabla.

IS NULL: La palabra clave IS NULL se utiliza en una condición de búsqueda para seleccionar filas que tienen un valor nulo en una columna especificada. Por ejemplo, para buscar los títulos de los libros en la tabla *titles* que no tienen datos en la columna *notes* (esto es, el valor de *notes* es *NULL)* se puede realizar la siguiente consulta:

```
SELECT    title,notes
FROM      titles
WHERE     notes IS NULL
GO
```

El resultado será el siguiente:

```
title                                      notes
-----------------------------------------  -------------
e Psychology of Computer Cooking           NULL
```

Como se puede ver, el valor nulo en la columna *notes* aparece como *NULL* en el resultado. *NULL* no es el valor real de la columna (simplemente indica que hay un valor nulo en esa columna).

Para encontrar los títulos que tienen datos en la columna *notes* (títulos para los cuales el valor es no es un valor nulo) hay que utilizar IS NOT NULL, según sigue:

```
SELECT    title,notes
FROM      titles
WHERE     notes IS NOT NULL
GO
```

Las 17 columnas de resultado tendrán uno o más caracteres en la columna *notes* y por consiguiente no tienen valores nulos en la columna *notes.*

IN: La palabra clave IN se utiliza en una condición de búsqueda para determinar si la expresión dada coincide con algún valor en una subconsulta o lista de valores. Si se encuentra una coincidencia se devuelve un valor *TRUE.* NOT IN devuelve la negación del resultado para IN y, por consiguiente, si no se encuentra la expresión en la subconsulta o en la lista de valores, se devuelve *TRUE.* La sintaxis es la siguiente:

<expresión de comprobación>IN «subconsulta »

<expresión de comprobación>IN «lista de valores»

Una *subconsulta* es una instrucción SELECT que devuelve solamente una columna en el resultado. La subconsulta debe estar entre paréntesis. Una *lista de valores* es justamente eso, con los valores entre paréntesis y separados por comas. La columna resultante de una subconsulta o de una lista de valores debe tener el mismo tipo de datos que *expresión de comprobación.* SQL Server realizará una conversión implícita cuando sea necesario. Se puede utilizar IN con una lista de valores para buscar los números ID de trabajos de tres descripciones de puestos de trabajo específicos, como se muestra en la siguiente consulta:

```
SELECT    job_id
FROM      jobs
WHERE     job_desc IN ("Operations Manager","Marketing Manager","Designer")
GO
```

La lista de valores en esta consulta es la siguiente: *(«Operations Manager»,«Marketing Manager»,«Designer»).* La consulta devuelve los números ID de trabajos de las consultas que tienen uno de estos tres valores en la columna *job_desc.* La palabra clave IN hace que la consulta sea más sirnple y sencilla de leer y entender que si se hubieran utilizado dos operadores OR, como se muestra seguidamente:

```
SELECT    job_id
FROM      jobs
WHERE     job_desc ="Operations Manager" OR
          job_desc ="Marketing Manager" OR
          job_desc ="Designer"
GO
```

La siguiente consulta utiliza la palabra clave IN dos veces en una instrucción (una vez en una subconsulta y otra vez en una lista de valores dentro de la subconsulta):

```
SELECT    fname, lname     --Outer query
FROM      employee
WHERE     job_id IN (      SELECT    job_id --consulta interior o subconsulta
                           FROM      jobs
                           WHERE     job_desc IN ("Operations Manager",
                                     "Marketing Manager", "Designer" ) )
GO
```

El resultado de la subconsulta se busca en primer lugar (en este caso un conjunto de valores *job_id).* Los valores *job_id* resultantes de la subconsulta no se muestran en la pantalla; la consulta externa los utiliza como la expresión de su propia condición de búsqueda IN. El resultado final contendrá los nombres y apellidos de todos los

empleados cuyos puestos de trabajos sean Operations, Manager, Marketing Manager o Designer. Veamos el resultado:

```
fname        lname
-----------  ----------------
Pedro        Afonso
Lesley       Brown
Palle        Ibsen
Karin        Josephs
Maria        Larsson
Elizabeth    Lincoln
Patricia     McKenna
Roland       Mendel
Helvetius    Nagy
Miguel       Paolino
Daniel       Tonini
(11 filas afectadas)
```

IN también se puede utilizar con el operador NOT. Por ejemplo, para obtener los nombres de todos los editores que no están ubicados en California, Texas o Illinois, se puede realizar la siguiente consulta:

```
SELECT    pub_name
FROM      publishers
WHERE     state NOT IN ( "CA", "TX", "IL")
GO
```

Esta consulta devolverá cinco filas donde el valor de la columna *state* no es uno de los tres estados en la lista de valores. Si ha establecido la opción *ANSI nulls* de la base de datos a *ON* el resultado contendrá solamente tres filas. Esta reducción es debida a que dos de las cinco columnas del resultado original tendrán un *NULL* en el valor de *state* y no se seleccionan los valores *NULL* cuando *ANSI nulls* se establece a ON.

Para determinar la configuración de *ANSI nulls* para la base de datos pubs, hay que ejecutar el siguiente procedimiento almacenado:

```
sp_dboption “pubs”, "ANSI nulls"
GO
```

Si ANSI nulls es *OFF*, se puede cambiar el valor a *ON* utilizando la siguiente instrucción:

```
sp_dboption “pubs”, "ANSI nulls",TRUE
GO
```

Para cambiar el valor de *ON* a *OFF*, hay que utilizar *FALSE* en lugar de *TRUE*.

Nota: Para más información sobre los efectos de la opción ANSI nulls de la base de datos, se puede consultar en el índice de los Libros en pantalla de SQL Server (SQL Server Books Online) «sp_dboption», y seleccionar «sp_dboption (T-SQL)» en el cuadro de diálogo Temas encontrados. También se puede consultar en el índice de los Libros en pantalla de SQL Server (SQL Server Books Online) «ANSI nufis» y entonces

pulsar sobre el vínculo «SET ANSI_NULLS» al final de la página para obtener el tema «SET ANSI_NULL (T-SQL)».

EXISTS: La palabra clave EXISTS se utiliza para verificar la existencia de filas en la subconsulta. La sintaxis es la siguiente:

EXISTS «subconsulta »

Si alguna fila satisface la subconsulta, se devuelve *TRUE.*

Para seleccionar nombres de autores que hayan publicado un libro, se puede utilizar la siguiente subconsulta:

```
SELECT    au_fname,au_lname
FROM      authors
WHERE     EXISTS (SELECT    au_id
                  FROM      titleauthor
                  WHERE     titleauthor.au_id =authors.au_id)
GO
```

No se seleccionarán los autores cuyos nombres estén en la tabla *authors* pero que no hayan publicado un libro, listado en la tabla *titleauthor*. Si no se habían seleccionado ninguna fila en la subconsulta, el resultado para la consulta exterior estará vacío (no se seleccionarán ninguna fila).

CONTAINS y FREETEXT: Las palabras clave CONTAINS y FREETEXT se utilizan para la búsqueda de texto completo en columnas del tipo de datos basadas en caracteres. Permite mayor flexibilidad a la palabra clave LlKE. Por ejemplo la palabra clave CONTAINS permite buscar las palabras que no coincidan exactamente, pero que son similares a la palabra o frase (una coincidencia «borrosa»). FREETEXT permite buscar palabras que coincidan o (coincidan borrosamente) parte o toda la cadena de búsqueda. Las palabras no tienen que coincidir con toda cadena de búsqueda, no tienen que estar en el mismo orden que las palabras en la cadena. Estas dos palabras clave se puede utilizar de muchas formas y ofrecen una serie de opciones relacionadas con búsquedas de texto completo.

CLÁUSULA GROUP BY

GROUP BY se utiliza después de la cláusula WHERE para indicar que las filas en el resultado deben estar agrupadas según las columnas de agrupación especificadas. Sí se utiliza una función agregado en la cláusula SELECT, se calcula un valor resumen de agregado para cada grupo y muestra en la salida. (Una función de agregado ejecuta un cálculo y devuelve un valor; estas funciones se describen detalladamente en la sección «Funciones de agregado» en este capítulo.)

> *Nota: Cada columna en la lista de selección (excepto las columnas utílizadas en una función de agregado) se tiene que especificar en la cláusula GROUP BY como una columna de agregación; si no es así SQL devolverá un mensaje de error. La salida podría no presentarse de una manera lógica si no se impone esta regla, puesto que columna GROUP BY especificada debe agrupar cada columna en la lista seleccionada.*

GROUP BY es muy útil cuando se incluye una función de agregado en la cláusula SELECT. Veamos la instrucción SELECT que utiliza la cláusula GROUP BY para buscar el número total de ventas de cada libro:

```
SELECT      title_id, SUM(qty)
FROM        sales
GROUP BY    title_id
GO
```

El resultado es el siguiente.

```
title_id
--------------      ----------
BU1032              15
BU1111              25
BU2075              35
BU7832              15
MC2222              10
MC3021              40
PC1035              30
PC8888              50
PS1372              20
PS2091              108
PS2106              25
PS3333              15
PS7777              25
TC3218              40
TC4203              20
TC7777              20
(16 filas afectadas)
```

Esta consulta no contiene una cláusula WHERE (no se necesita). El resultado muestra una columna *title_id* y una columna resumen sin cabecera. Para cada identificador de título distinto aparece el número total de ventas del título en la columna resumen. Por ejemplo, el valor *title_id* BU1032 aparece dos veces en la tabla *sales* (aparece una vez mostrando 5 en la columna qty y aparece de nuevo mostrando 10 para un orden diferente. La función de agregado SUM suma estas dos ventas para llegar al total de ventas de 15, la cual aparece en la columna resumen.

Para agregar una cabecera a la columna resumen, se puede utilizar la palabra clave AS, como muestra en el siguiente ejemplo:

```
SELECT      title_id,SUM(qty) AS "Total Sales"
FROM        sales
GROUP BY    titie_id
GO
```

Ahora el resultado mostrará la cabecera «Total Sales» encima de la columna resumen:

```
title_id   Total Sales
---------- -----------
BU1032     15
BU1111     25
BU207 5    35
BU7832     15
MC2222     10
MC3021     40
PC1035     30
PC8888     50
PS1372     20
PS2091     108
PS2106     25
PS3333     15
PS7777     25
TC3218     40
TC4203     20
TC7777     20
(16filas afectadas)
```

Se pueden anidar grupos que incluyen más de una columna en la cláusula GROUP BY. Anidar grupos significa que el resultado se agrupará por cada una de las columnas de agrupación en el orden en el que se especifican las columnas. Por ejemplo, para encontrar el precio medio de los libros que están agrupados por el tipo y seguidamente por el editor se ha de realizar la siguiente consulta: Se pueden anidar grupos incluyendo más de una columna en la cláusula GROUP BY.

```
SELECT      type,pub_id,AVG(price)AS "Average Price"
FROM        titles
GROUP BY    type,pub_id
GO
```

El resultado es el siguiente:

```
type          pub-id       Average Price
----------    ----------   ----------------------
business      0736         2.99
psychology    0736         11.48
UNDECIDED     0877         NULL
mod_cook      0877         11.49
psychology    0877         21.59
trad cook     0877         15.96
business      1389         17.31
popular_comp  1389         21.48
(8 filas afectadas)
```

Nótese que los tipos psychology y business ocurren más de una vez puesto que están agrupados bajo un identificador de editor diferente. El precio medio *NULL* para el tipo UNDECIDED refleja que no se introdujeron precios en la tabla para este tipo y por consiguiente no se pudo calcular ningún promedio.

GROUP BY proporciona una palabra clave opcional, ALL, que especifica que se tienen que incluir todos los grupos en el resultado, incluso si no cumplen la condición de búsqueda. Los grupos que no tienen filas que cumplan la condición de búsqueda contendrán *NULL* en la columna resumen, de forma que se pueden identificar fácilmente. Por ejemplo, para mostrar que el precio medio de los libros que tienen unos derechos de autor de un 12 por ciento (y también mostrar los libros que no, los cuales tendrán *NULL* en la columna resumen). Y para agrupar los libros por tipo y posteriormente por identificador de editor se puede ejecutar la siguiente consulta:

```
SELECT      type,pub_id,AVG(price) AS "Average Price"
FROM        titles
WHERE       royalty =12
GROUP BY ALL type,pub_id
GO
```

El resultado será el siguiente:

```
type          pub_id          Average Price
----------    ----------      ---------------------
business      0736            NULL
psychology    0736            10.95
UNDECIDED     0877            NULL
mod_cook      0877            19.99
psychology    0877            NULL
trad_cook     0877            NULL
business      1389            NULL
popular_comp  1389            NULL
(8 filas afectadas)
```

En la salida están presentes todos los tipos y aparece *NULL* para los tipos que no tienen un libro con una comisión de un 12 por ciento.

Si quitamos la palabra clave ALL, el resultado contendrá solamente los tipos que tienen un libro con una comisión de un 12 por ciento, como se muestra seguidamente:

```
type          pub_id          Average Price
----------    ----------      --------------------
psychology    0736            10.95
mod_cook      0877            19.99
(2 filas afectadas)
```

La cláusula GROUP BY frecuentemente viene acompañada de la cláusula HAVING.

CLÁUSULA HAVING

Se utiliza la cláusula HAVING par especificar una condición de búsqueda para un grupo o función de agregado. HAVING se utiliza normalmente después de una cláusula GROUP BY para casos en los cuales se tiene que verificar una condición de búsqueda después de que se agrupen los resultados. Si se puede aplicar la condición de búsqueda antes de que ocurra el agrupamiento, es más eficiente ubicar la condición de búsqueda en la cláusula WHERE que agregar una cláusula HAVING. Esta técnica reduce el número de filas que se tienen que agrupar. Si no hay ninguna cláusula GROUP BY, se puede utilizar HAVING solamente con una función de agregado en la lista de selección. En este caso, la cláusula HAVING actúa de la misma forma que la cláusula WHERE. Si no se utiliza HAVING de alguna de estas formas, SQL Server devolverá un mensaje de error.

La sintaxis para la cláusula HAVING es la siguiente:

HAVING <condición de búsqueda>

Aquí, *condición de búsqueda* tiene el mismo significado que las condiciones de búsqueda descritas en la sección «Cláusula WHERE y condiciones de búsqueda» de este capítulo. Una diferencia entre la cláusula HAVING y la cláusula WHERE es que la cláusula HAVING puede incluir una función de agregado en la condición de búsqueda, mientras que la cláusula WHERE no. Es decir, se pueden utilizar funciones de agregado en la cláusula SELECT y en la cláusula HAVING, pero no se pueden utilizar en la cláusula WHERE.

La siguiente utiliza una cláusula HAVING para seleccionar los tipos de libros por editor que un precio medio mayor que 15:

```
SELECT      type,pub_id,AVG(price) AS “Average Price"
FROM        titles
GROUP BY    type,pub_id
HAVING      AVG(Price) >15.00
GO
```

El resultado es el siguiente:

```
type          pub_id      Average Price
--------------- ----------- ----------------------
psychology    0877              21.59
trad_cook     0877              15.96
business      1389              17.31
popular_comp  1389              21.48
(4 filas afectadas)
```

También se pueden utilizar operadores lógicos con la cláusula HAVING. En la siguiente consulta se ha agregado el operador AND:

```
SELECT      type,pub_id,AVG(price) AS "Average Price"
FROM        titles
GROUP BY    type,pub_id
```

```
HAVING      AVG(price) >=15.00 AND
            AVG(price)<=20.00
GO
```

El resultado es el siguiente

```
type          pub_id  Average Price
------------  ------  ----------------
trad cook     0877          15.96
business      1389          17.31
(2 filas afectadas)
```

Se pueden obtener los mismos resultados utilizando la cláusula BETWEEN en lugar de AND, como se muestra seguidamente:

```
SELECT      type,pub_id,AVG(price) AS "Average Price"
FROM        titles
GROUP BY    type,pub_id
HAVING      AVG(price) BETWEEN 15.00 AND 20.00
GO
```

Para utilizar HAVING sin una cláusula GROUP BY, se debe tener una función de agregado en la lista de selección y en la cláusula HAVING. Por ejemplo, para seleccionar la suma de los precios de los libros del tipo *mod_cook,* solamente si la suma es mayor a 20, se ha de ejecutar la siguiente consulta:

```
SELECT      SUM(prices)
FROM        titles
WHERE       type ="mod_cook"
HAVING      SUM(price)>20
GO
```

Si se intenta introducir *SUM(price)>20* en la cláusula WHERE, SQL Server devolverá un rnensaje de error (las funciones de agregado no están permitidas en la cláusula WHERE). Hay que recordar que la única vez que se puede utilizar cláusula HAVING es cuando se agrega una condición de búsqueda para verificarlos grupos resultantes de una cláusula GROUP BY o para verificar una función de agregado. En otro caso, se debería especificar la condición de búsqueda en la cláusula WHERE.

CLÁUSULA ORDER BY

Se utiliza la cláusula ORDER BY para especificar el orden en el que se han de ordenar las filas en un resultado. Se puede especificar tanto en orden ascendente (de menor a mayor) o descendente (de mayor a menor) mediante el uso de ASC o DESC. Si no se especifica, el orden predeterminado es el orden ascendente. Se puede especificar más de una columna en la cláusula ORDER BY. El resultado se ordenará según la primera columna listada. Si la primera columna contiene valores duplicados, estas filas se ordenarán según la segunda columna y así sucesivamente. Esta ordenación tiene más sentido cuando se utiliza ORDER BY con GROUP BY, como se verá posteriormente en esta sección. Primeramente veamos un ejemplo que utiliza una columna en la cláusula ORDER BY para listar los autores por su apellido, en orden ascendente:

```
SELECT      au_lname,au,_fname
```

```
FROM        authors
ORDER BY    au_lname ASC
GO
```

El resultado estará ordenado alfabéticamente por el apellido. Hay que recordar que la distinción de mayúsculas en el orden que se establece cuando se instala SQL Server afectará a cómo se ordenan los apellidos tales como «del Castillo». Si se desea ordenar los resultados según más de una columna, simplemente hay que agregar los nombres de las columnas, separados por comas, a la cláusula ORDER BY La siguiente columna selecciona los identificadores de los puestos de trabajo y los nombres y apellidos de los empleados los muestra ordenados por el identificador del puesto de trabajo, el apellido y el nombre:

```
SELECT      job_id,lname,fname
FROM        employee
ORDER BY    job_id,lname,fname
GO
```

job_id	lname	fname
2	Cramer	Philip
3	Devon	Ann
4	Chang	Francisco
5	Henríot	Paul
5	Hernadez	Carlos
5	Labrune	Janine
5	Lebihan	Laurence
5	Muller	Rita
5	Ottlieb	Sven
5	Pontes	Maria
6	Ashworth	Victoria
6	Karttunen	Matti
6	Roel	Diego
6	Roulet	Annette
7	Brown	Lesley
7	Ibsen	Palle
7	Larsson	Maria
7	Nagy	Helvetius
...	...	...
13	Accorti	Paolo
13	O'Rourke	Timothy
13	Schmitt	Carine
14	Afonso	Pedro
14	Josephs	Karin
14	Linco1n	Elizabeth

(43 filas afectadas)

El orden de los nombres en la columna no afectará al resultado puesto que no existen dos empleados que tengan el mismo apellido y el mismo identificador del puesto de trabajo.

Veamos ahora una cláusula ORDER BY con una cláusula GROUP BY y una función de agregado:

```
SELECT      type,pub_id,AVG(price) AS "Average Price"
FROM        titles
GROUP BY    type,pub_id
ORDER BY    type
```

```
GO
```

El resultado será:

```
type                 pub_id  Average Price
-------------------- ------- ------------------
UNDECIDED            0877    NULL
business             0736    2.99
business             1389    17.31
mod_cook             0877    11.49
popular_comp         1389    21.48
psychology           0736    11.48
psychology           0877    21.59
trad_cook            0877    15.96
(8 filas afectadas)
```

Los resultados se ordenan alfabéticamente (en orden ascendente) según *type.* También, nótese que en esta consulta tanto *type.* como *pub_id* deben estar en la cláusula GROUP BY puesto que no son parte de una función de agregado. Si se hubiera omitido la columna *pub_id* de la cláusula GROUP BY, SQL Server habría mostrado un mensaje de error.

No se pueden utilizar las funciones de agregado o subconsultas en la cláusula ORDER BY. Sin embargo, si se había dado un alias a un agregado en la cláusula SELECT se podía utilizar en la cláusula ORDER BY, como se muestra seguidamente:

```
SELECT      type,pubid,AVG(price) AS "Average Price"
FROM        titles
GROUP BY    type,pub_id
ORDER BY    "Average Price"
GO
```

El resultado será:

```
type                 pub_id  Average Price
-------------------- ------- --------------------
UNDECIDED            0877    NULL
business             0736    2.99
psychology           0736    11.48
mod_cook             0877    11.49
psychology           0877    21.59
trad_cook            0877    15.96
business             1389    17.31
popular_comp         1389    21.48
(8 filas afectadas)
```

Ahora los resultados están ordenados según el precio medio. *NULL* se considera el primero en la ordenación, por lo que está al principio de la lista.

OPERADOR UNION

UNION es considerado como un operador, en lugar de cómo una cláusula. Se utiliza para combinar los resultados de dos o más consultas en un resultado. Se deben seguir dos reglas cuando se utiliza UNION:

- El número de columnas debe ser el mismo en todas las consultas.

- Los tipos de datos para columnas correspondientes deben ser compatibles.

Las columnas listadas en las instrucciones SELECT unidas por UNION se corresponden de la siguiente manera: la primera columna de la primera instrucción SELECT corresponderá a la primera columna en cada instrucción SELECT subsiguiente, la segunda columna corresponderá a la segunda columna en la instrucción SELECT subsiguiente, y así sucesivamente. Por consiguiente, se debe tener el mismo número de columnas en todas las instrucciones SELECT unidas por UNION para asegurar una correspondencia uno a uno. Además, las columnas correspondientes deben tener tipos de datos compatibles, por lo que dos columnas correspondientes deben tener el mismo tipo de datos o SQL Server debe poder convertir de forma implícita un tipo de datos en otro. El siguiente ejemplo utiliza UNION para unir los resultados de dos instrucciones SELECT que recuperan las columnas city y s*tate* de las tablas *publishers y stores:*

```
SELECT    city,state
FROM      publishers
UNION
SELECT    city,state
FROM      stores
GO
```

El resultado obtenido es:

```
city            state
--------------  ----------
Fremont         CA
Los Gatos       CA
Portland        OR
Remulade        WA
Seattle         WA
Tustin          CA
Chicago         IL
Dallas          TX
München         NULL
Boston          MA
New York        NY
Paris           NULL
Berkeley        CA
Washington      DC
(14 filas afectadas)
```

Las dos columnas, *city* y *state* son del mismo tipo de datos (char) en ambas tablas *publishers* y *stores*; por consiguiente no se necesita conversión de los tipos de datos. Las cabeceras de las columnas para el resultado de UNION se toman de la primera instrucción SELECT. Si se desea crear un alias para una cabecera, hay que ponerlo en la primera instrucción SELECT, como se muestra seguidamente mente:

```
SELECT    city AS "All Cities", state AS "All States"
FROM      publishers
UNION
SELECT    city,state
FROM      stores
GO
```

El resultado será:

```
All Cities     All States
---------------  --------------
Fremont        CA
Los Gatos      CA
Portland       OR
Remulade       WA
Seattle        WA
Tustin         CA
Chicago        IL
Dallas         TX
München        NULL
Boston         MA
New York       NY
Paris          NULL
Berkeley       CA
Washington     DC
(14 filas afectadas)
```

No es necesario utilizar las mismas columnas en ambas cláusulas SELECT en una unión. Por ejemplo, se podía haber seleccionado *city* y s*tate* de la tabla *store y* seleccionar *city* y *country* de la tabla *publishers* como se muestra seguidamente:

```
SELECT    city,country
FROM      publishers
UNION
SELECT    city,state
FROM      stores
GO
```

El resultado es el siguiente:

```
city           country
---------------  -------------
Fremont        CA
Los Gatos      CA
Portland       OR
Remulade       WA
Seattle        WA
Tustin         CA
```

```
New York        USA
Paris           France
Boston          USA
München         Germany
Washington      USA
Chicago         USA
Berkeley        USA
Dallas          USA
(14 filas afectadas)
```

En este resultado, las seis primeras filas provienen de la consulta en la tabla *stores* y las últimas ocho filas son los resultados de la consulta en la tabla *publishers.* La columna *state* tiene el tipo de datos *char y la* columna *country* tiene el tipo de datos *varchar*. Puesto que estos tipos de datos son compatibles, SQL Server realizará una conversión implícita de forma que ambas columnas pasan al tipo de datos *varchar.* Las cabeceras de columna son *city* y *country,* puesto que viene de las columnas listadas en la primera instrucción SELECT, pero en este resultado una mejor elección para la cabecera de la columna *country* sería «State or Country».

Se puede utilizar una palabra clave con UNION: ALL. Si se especifica ALL, se incluirán filas duplicadas en el resultado (en otras palabras, se incluirán todas las filas). Si no se especifica ALL de forma predeterminada se eliminan las filas duplicadas del resultado.

Se puede utilizar ORDER BY solamente en la última instrucción de una unión, no en cada instrucción SELECT. Esta limitación asegura que el resultado final estará ordenado una vez para todos los resultados. Por otra parte, se puede utilizar GROUP BY y HAVING en instrucciones individuales, puesto que afectarán solamente a los resultados individuales y no al resultado final. La siguiente unión junta los resultados de dos instrucciones SELECT que tienen una cláusula GROUP BY.

```
SELECT      type,COUNT(title) AS "Number of Titles"
FROM        titles
GROUP BY    type
UNION
SELECT      pub_name,COUNT(titles.title)
FROM        publishers,titles
WHERE       publishers.pub_id =titles.pub_id
GROUP BY    pub_name
GO
```

El resultado es:

```
type                        Number of Titles
--------------------------  -------------------------
Algodata Infosystems        6
Binnet &Hardley             7
New Moon Books              5
psychology                  5
mod_cook                    2
trad_cook                   3
popular_comp                3
UNDECIDED                   1
business                    4
(9 filas afectadas)
```

Este resultado muestra cuántos títulos ha publicado cada editor que ha publicado títulos (las primeras tres filas del resultado) y muestra el número de títulos en cada categoría. Cada GROUP BY se ejecutó en cada subconsulta.

El operador UNION se puede también utilizar con más de dos instrucciones SELECT. Hay que tener cuidado cuando se crea una unión de asegurar que todas las columnas y tipos de datos en las consultas se corresponden correctamente.

FUNCIONES DE AGREGADO DE T-SQL

Ahora que se está familiarizando con las cláusulas básicas que se pueden utilizar con la instrucción SELECT, veamos algunas funciones T-SQL que se pueden utilizar en la cláusula SELECT para tener mayor flexibilidad en la creación de consultas. Estas funciones están agrupadas en varias categorías, incluyendo funciones de configuración, de cursor, de fecha y hora, de seguridad, de metaarchivos, del sistema, estadísticas del sistema, texto e imagen, matemáticas, conjunto de filas, cadena de caracteres y de agregados. Estas funciones pueden realizar cálculos, conversiones y otras operaciones o pueden devolver cierta información. Hay muchas funciones disponibles, pero en esta sección solamente examinaremos las funciones de agregado.

Como hemos señalado, una función de agregado realiza un cálculo sobre un conjunto de valores y devuelve un único valor. Las funciones de agregado se pueden especificar en la lista de selección y se usan frecuentemente cuando la instrucción contiene una cláusula GROUP BY. En algunos de los ejemplos anteriores se utilizaron las funciones AVG y COUNT. La tabla siguiente lista las funciones de agregado disponibles:

Función	Descripción
AVG	Devuelve el promedio de los valores en la expresión; ignora todos los valores nulos.
COUNT	Devuelve el número de elementos (igual para el número de filas) en una expresión.
COUNT_BIG	Lo mismo que COUNT, excepto que devuelve la cuenta en el tipo de datos bigint, en lugar de en int.
MAX	Devuelve el valor máximo de una expresión.
MIN	Devuelve el valor mínimo de una expresión.
STDEV fun-	Devuelve la desviación estándar estadística de todos los valores en una expresión. La ción asume que los valores utilizados en el cálculo son una muestra de la población completa
STDEVP utiliza	Devuelve la desviación estándar estadística para la población de todos los valores dos en una expresión. Esta función asume que los valores utilizados en el cálculo constituyen toda la población.
SUM	Devuelve la suma de todos los valores en una expresión.

VAR	Devuelve la varianza estadística de todos los valores en la expresión. Esta función asume que los valores utilizados en el cálculo son una muestra de la población completa.
VARP	Devuelve la varianza estadística para la población de todos los valores en una expresión. Esta función asume que todos los valores utilizados en el cálculo constituyen toda la población.

La función COUNT tiene un uso especial: contar todas las filas en una tabla. Para realizar esto, se ha de utilizar después de COUNT, como se muestra seguidamente:

```
SELECT    COUNT(*)
FROM      publishers
GO
```

El resultado será el siguiente:

```
-----------
      8
```

Este resultado indica que la tabla *publishers* contiene ocho filas. Las funciones AVG, COUNT, MAX, MIN y SUM se pueden utilizar con las palabras claves opcionales ALL y DISTINCT. Para cada una de estas funciones ALL indica que la función se debe aplicar a todos los valores en la expresión y DISTINCT indica que los valores duplicados solamente se deben contar una vez en los cálculos. La opción predeterminada es ALL.

Las funciones de agregado son básicamente autoexplicativas. El siguiente ejemplo utiliza las funciones MAX y MIN con el fin de encontrar la diferencia de precio entre el libro más caro y el más barato:

```
SELECT    MAX(price) - MIN((price) AS "Price Difference"
FROM      titles
GO
```

El resultado será:

```
Price Difference
----------------------
19.96
```

En el siguiente ejemplo se utiliza SUM para encontrar la cantidad total de elementos ordenados por tienda:

```
SELECT    stores.stor_name,SUM(sales.qty) AS "Total Items Ordered"
FROM      sales,stores
WHERE     sales.stor_id =stores.stor_id
GROUP BY  stor_name
GO
```

El resultado será:

stor_name	*Total Items Ordered*
Barnum's	*125*
Bookbeat	*80*
Doc-U-Mat:Quality Laundry and Books	*130*
Eric the Read Books	*8*
Fricative Bookshop	*60*
News &Brews	*90*

(6 filas afectadas)

LA INSTRUCCIÓN INSERT

La instrucción INSERT se utiliza para añadir nuevas filas tabla o vista. La sintaxis básica de la instrucción INSERT se muestra a continuación:

```
INSERT [INTO] nombre_de_ tabla [ (lista_de_columnas) ] VALUES expresión |
tabla_derivada
```

La palabra clave INTO y el parámetro *lista_de_columnas* son opcionales. El parametro *lista_de_columnas* especifica las columnas a las que se añaden datos; estos valores tendrán una correspondencia biunívoca (en su orden) con los que aparecen en la expresión (que puede ser simplemente una lista de valores). Se examinarán algunos ejemplos.

Añadido de filas

Antes de comenzar a trabajar se creará una tabla, *items,* para emplearla en los ejemplos. Los comandos de T-SQL empleados para la tabla items se muestran a continuación:

```
USE MyDB
GO
CREATE TABLE items
(
item_category  CHAR(20)        NOT NULL,
item_id                SMALLINT        NOT NULL,
price                  SMALLMONEY  NULL,
item_desc              VARCHAR(30)   DEFAULT 'No desc'
)
GO
```

El siguiente ejemplo de código muestra la manera de añadir una única fila de datos a tabla *items:*

```
INSERT INTO  items
             (item category,   item_id, price,    item desc)
VALUES       ('health food',   1,       4.00,     'tofu 6 oz.')
GO
```

Como se ha especificado un valor para cada columna de la tabla y se ha hecho referencia a esos valores en el mismo orden en el que se han definido en la tabla las columnas correspondientes, no se utiliza el parámetro *lista_de_columnas.* Pero si los valores no estuvieran en el mismo orden que las columnas, se podría introducir un dato incorrecto en una columna o recibir un mensaje de error.

Por ejemplo, si se intenta ejecutar la siguiente instrucción, se obtendrá el mensaje de error que aparece a continuación:

```
INSERT INTO  items
VALUES       (1, 'health food',  4. 00, ' tofu 6 oz.')
GO

Servidor: mensaje 245, nivel 16, estado 1, línea 1
```

Error de sintaxis al convertir el valor varchar 'health food ' para una
Columna de tipo de datos smallint.

Se obtiene este mensaje y no se añade la fila porque los valores se han ordenado de manera incorrecta. Se ha intentado añadir el elemento *ID* en la columna *item_category*, y el elemento *category* en la columna *item_id.* Los valores no eran compatibles con los tipos de datos de esas columnas. Si hubieran sido compatibles, SQL Server hubiera permitido que se añadiera la fila, aunque los valores no estuvieran en el lugar adecuado.

Para ver cómo aparece la fila añadida a la tabla se solicita a la tabla que seleccione todas las filas empleando la siguiente instrucción SELECT:

```
SELECT * from items
GO
```

Se obtendrá el siguiente resultado:

```
Item_category  item_id        price      item_desc
-------------- -------------- ---------- -------------
health food    1              4.00       tofu 6 oz.
```

Al crear la tabla *items* se definió la columna *price* para que permitiera valores null, y se asigna la columna *item_desc* (descripción) un valor predeterminado de *No desc.* Si no se especifica ningún valor para la columna *price* en la instrucción INSERT se le añade *NULL* en la fila nueva. Si no se especifica ningún valor para la columna *item_desc* se le añade el valor predeterminado *No desc* en la fila nueva.

Omisión de los valores de las columnas

En la primera instrucción INSERT de ejemplo del apartado precedente se podrían haber omitido los valores y los nombres de columna de las columnas *price* e *item_desc* ya que ambas columnas tienen valores predeterminados. Si se omite el valor de una columna hay que especificar el resto de columnas en *lista_de_columnas* ya que, de otro modo, SQL Server asignará los valores indicados a las columnas en el orden en que éstas se definieron en la tabla.

Por ejemplo, supóngase que se excluye el valor de la columna *price* y no se especifica ningún valor para *lista_de_columnas* como en esta consulta:

```
INSERT INTO  items
VALUES       ('junk food',2,'fried pork skins')
GO
```

SQL Server intentará añadir el valor dado para *item_desc (fried pork skins;* el tercer valor de la lista de valores) a la columna *price* (la tercera columna de la tabla). Se producirá un mensaje de error, ya que *fried pork skins* es un valor con tipo de datos *char*, mientras que *price* tiene el tipo de datos *smallmoney*. Se trata de tipos de datos incompatibles. El mensaje de error tendrá un aspecto similar a éste:

Servidor: mensaje 213, nivel 16, estado 4, línea 1
Error de inserción: el nombre de columna o los valores especificados no corresponden a la definición de la tabla.

Se puede imaginar lo que podría haber ocurrido a la integridad de la tabla si *fried pork skins* hubiera sido un valor con un tipo de datos compatible con el especificado para *price.* SQL Server habría añadido sin saberlo el valor a una columna que no le corresponde y la tabla hubiera tenido datos inconsistentes.

Hay que recordar que los valores añadidos a una tabla o a una vista deben ser de un tipo de datos compatible con la definición de la columna correspondiente. Además, si la fila que se añade viola una regla o una restricción, se recibirá un mensaje de error de SQL Server, y la adición fallará.

Para evitar errores de tipo de datos incompatible hay que ordenar los nombres de *lista_de_columnas* para que coincidan con el de sus valores correspondientes, como se muestra a continuación:

```
INSERT INTO items (item_category,item_id,item_desc)
VALUES  ('junk food ',2,'fried pork skins')
GO
```

Como no se ha especificado el precio, la columna *price* tiene un valor NULL en esta fila. Ahora se ejecutará la siguiente instrucción SELECT

```
SELECT *FROM items
```

Se obtendrá el siguiente conjunto de resultados (que ahora incluye las dos filas que se han añadido). Obsérvese el *NULL* en la columna *price.*

```
item_category   item_id     price          item_desc
--------------------        -----------    -----------      ------------------
health food     1           4.00           tofu 6 oz.
junk food       2           NULL           fried pork skins
```

Ahora se añadirá otra fila, sin especificar los valores de las columnas *price* ni *item_desc,* como se muestra a continuación:

```
INSERT INTO items  (item_category,item id)
VALUES ('toys',3)
GO
```

El conjunto de resultados de esta fila puede hallarse empleando esta consulta:

```
SELECT *FROM items
WHERE item_id =3
```

El conjunto de resultados tendrá el aspecto siguiente:

```
item_category   item_id       price          item_desc
------------------          -----------     ----------       ----------------
toys            3               NULL        No desc.
```

Obsérvense los valores *NULL* de la columna *price* y *No dese* de la columna *item_desc* Estos valores pueden modificarse empleando la instrucción UPDATE, como se verá más adelante en este capítulo.

Los cuatro tipos de columna para las que SQL Server proporciona de manera automática un valor cuando no se especifica ninguno son las columnas que permiten valores null, las que tienen un valor predeterminado, las columnas de identidad y las columnas de *timestamp.* Ya se ha visto lo que ocurre con las columnas que permiten valores null y con las columnas que tienen valores predeterminados. Las columnas de identidad obtiene el siguiente valor de identidad que esté disponible y las columnas de *timestamp* obtienen el valor actual de la timestamp. En la mayor parte de los casos no se pueden añadir valores de datos de manera manual a estos tipos de columna.

Adición de filas procedentes de otra tabla

También se pueden añadir a una tabla filas procedentes de otra. Esto se consigue empleando una tabla derivada en la instrucción INSERT o utilizando la cláusula EXECUTE con un procedimiento almacenado que devuelva filas de datos.

Una tabla derivada es el conjunto de resultados de una instrucción SELECT anidada en la cláusula FROM de otra instrucción de T-SQL. Para llevar a cabo una adición empleando una tabla derivada se va a crear previamente otra tabla de pequeño tamaño denominada *two_newest_items,* en la que se añadirán las filas procedentes de la tabla *items.* La instrucción CREATE TABLE de la nueva tabla se muestra a continuación:

```
CREATE TABLE        two_newest_items
(
item_id         SMALLINT        NOT NULL,
item_desc       VARCHAR(30)     DEFAULT 'No desc'
)
GO
```

Para añadir los dos valores de las columnas *item_id* e *item_desc* procedentes de la tabla *items* a la tabla *two newest_items* hay que emplear la siguiente instrucción INSERT:

```
INSERT INTO two_newest_items (item_id,item_desc)
SELECT TOP 2 (item_id,item_desc)
FROM items
ORDER BY item id DESC
GO
```

Hay que tener en cuenta que en lugar de utilizar una lista de valores en esta instrucción INSERT se ha empleado una instrucción SELECT. Esta instrucción SELECT devuelve datos una tabla ya existente que se utilizan como lista de valores. Además, hay que tener en cuenta la instrucción SELECT no se ha utilizado entre paréntesis, lo que haría que se produjera un mensaje de error de sintaxis. Se pueden consultar todas las filas de esta tabla nueva empleando esta consulta:

```
SELECT *
FROM two_newest_items
```

Aparecerá el siguiente conjunto de resultados:

```
item_id      - item_desc
-----------  - -------------------
3            - No desc
2            - fried pork skins
```

Hay que tener en cuenta que se ha incluido la cláusula ORDER BY *item_id* DESC en la instrucción INSERT. Esta cláusula indica a SQL Server que debe ordenar los resultado, decreciente de *item_id.*

LA INSTRUCCIÓN UPDATE

La instrucción UPDATE se utiliza para modificar o actualizar los datos existentes. La sintaxis básica UPDATE se muestra a continuación:

```
UPDATE nombre_de_tabla SET nombre_de_columna=expresión
[FROM origen_de_tabla] WHERE condición_de_búsqueda
```

Actualización de filas

Siguiendo con la tabla de ejemplo, *items*, primero se actualizará la fila *junk food* que se añadió anteriormente sin incluir el precio. Para identificar la fila hay que especificar *fried pork skins* en la condición de búsqueda. Para definir (actualizar) el precio como 2 hay que utilizar la instrucción siguiente:

```
UPDATE      items
SET         price =2.00
WHERE       item_desc = 'fried pork skins'
GO
```

Ahora se selecciona la fila *junk food* usando esta consulta:

```
SELECT  * FROM items
WHERE item_desc = 'fried pork skins'
GO
```

El resultado de la fila *junk food* tiene el siguiente aspecto, con el valor original NULL del precio sustituido por 2.00:

```
item_category  item_id     price          item_desc
-------------------        ------------  -----------  ------------------
food                       2             2.00         fried pork skins
```

Para aumentar el precio de este elemento un diez por ciento habría que ejecutar la siguiente instrucción

```
UPDATE      items
SET         price =price *1.10
WHERE       item_desc = ‘fried pork skins'
GO
```

Si ahora se selecciona la fila *junk food* se observará que el precio ha cambiado a 2.20 (2 multiplicados por 1. 10). Los precios de los demás elementos no han cambiado.

Se pueden actualizar varias filas empleando una instrucción UPDATE. Por ejemplo, para actualizar todas las filas de la tabla *items* aumentando los valores de *price* un diez por ciento, hay que ejecutar la instrucción siguiente:

```
UPDATE      items
```

```
SET         price =price *1.10
GO
```

Si se examina ahora la tabla *items*, tendrá un aspecto parecido a éste:

item_category	item_id	price	item_desc
health food	1	4.40	tofu 6 oz.
junk food	2	2.42	fried pork skins
toys	3	NULL	No desc

Las filas que tengan como *price* un valor *NULL* no se verán afectadas, ya que *NULL * 1.10= NULL*. No es ningún problema, no se recibirá ningún mensaje de error.

Uso de la cláusula FROM

La instrucción UPDATE permite utilizar la cláusula FROM para especificar una tabla que se utilizará como fuente de datos en una actualización. La lista de orígenes de la tabla puede incluir nombres de tablas, nombres de vistas, funciones de definición de filas, tablas derivadas y tablas combinadas. Se puede utilizar como origen de la tabla incluso la propia tabla que se está actualizando. Para ver el modo en que funciona este proceso antes hay que crear otra tabla de ejemplo de pequeño tamaño. A continuación se muestran la instrucción CREATE TABLE de la nueva tabla, denominada *tax*, y una instrucción INSERT para añadir una fila con el valor de 5.25 en la columna *tax_percent:*

```
CREATE TABLE tax
(           tax_percent:        real            NOT NULL,
            change_date         smalldatetime   DEFAULT getdate()
)
GO
INSERT INTO tax (tax_percent)
VALUES (5.25)
GO
```

La columna *change_date* recibe la fecha y la hora actuales de su función predeterminada GETDATE porque no se ha añadido de manera explícita ninguna fecha.

Ahora se añadirá una nueva columna que puede tener valores null, *price_with_tax*, a la tabla *items*, como se muestra a continuación:

```
ALTER TABLE items
ADD price_with_tax, smallmoney NULL
GO
```

Ahora se desea actualizar la nueva columna *price_with_ tax* para que contenga el resultado de *items.price *tax.tax_percent* en todas las filas de la tabla *items.* Para ello, se utiliza la siguiente instrucción UPDATE con una cláusula FROM:

```
UPDATE      items
SET         price_with_tax = i.price + (i.price *t.tax_percent /100)
FROM        items i,tax t
```

```
GO
```

Las dos filas de la tabla *items* que tienen un valor en la columna *price* tienen ahora un valor calculado en la columna *price_with_tax*, La fila que tenga un valor *NULL* de *price* no se ve afectada, y su valor de *price_with_tax*, es también *NULL*, porque un valor NULL multiplicado por cualquier cosa es *NULL*. El conjunto de resultados de todas las filas de la tabla *items* (incluidas todas las modificaciones realizadas hasta ahora) tiene este aspecto:

item category	item id	price	item desc	price_with_tax
health food	1	4.40	tofu 6 oz.	4.63
junk food	2	2.42	fried pork skins	2.55
toys	3	NULL	No desc	NULL

Uso de tablas derivadas

Otra manera de emplear la instrucción UPDATE, es hacerlo con una tabla derivada, o subconsulta, en la cláusula FROM. La tabla derivada se emplea como entrada de datos para la instrucción UPDATE exterior. Para este ejemplo se utilizará la tabla *two_newest_items* de la subconsulta y la tabla *items* de la instrucción UPDATE, exterior. Se desea actualizar las dos filas más recientes de la tabla *items* para que contengan un valor de *NULL* en *price_with_tax.* Al consultar la tabla *two_newest_items* se pueden encontrar los valores de *item_id* de las filas que hay que actualizar en la tabla *items.* Eso lo logra la siguiente instrucción:

```
UPDATE      items
SET         price with tax = NULL
FROM        (SELECT item_id FROM two_newest_items) AS t1
WHERE       items.item_id = t1.item_id
GO
```

La instrucción SELECT sirve de subconsulta, cuyos resultados se ubican en una tabla derivada temporal denominada *t1* que luego se utiliza en la condición de búsqueda (la cláusula WHERE). Los resultados de la subconsulta nos dan los valores de *item_id* 2 y 3. Así, se ven afectadas las dos filas de la tabla items con valores de 2 o 3 en la columna *item_id*. La fila con un valor item_id de 3 ya tenía un valor de *NULL* en la columna *price_with_tax*, por lo que no se modifican sus valores. La fila con un valor item_id de 2 ve modificado su valor de *price_with_tax*, modificado a *NULL*. El conjunto de resultados que muestra todas las filas de la tabla *items* después de su actualización tiene el aspecto que se muestra a continuación:

item_category	item_id	price	item_desc	price_with_tax,
health food	1	4.40	tofu 6 oz.	4.63
junk food	2	2.42	fried pork skins	NULL
toys	3	NULL	No desc	NULL

i

LA INSTRUCCIÓN DELETE

La instrucción DELETE se utiliza para eliminar (borrar) filas de una tabla o de una vista. DELETE no afecta a la definición de la tabla; sencillamente, le borra filas de datos. La sintaxis básica de la instrucción DELETE se muestra a continuación:

```
DELETE [FROM] nombre_de_tabla | nombre_de_vista
[FROM origenes_de_tabla] WHERE condición_de_búsqueda
```

La primera palabra clave FROM es opcional, igual que la segunda cláusula FROM. Las filas no se borran de los orígenes de la tabla de la segunda cláusula FROM; sólo se borran de vista especificada después de DELETE.

Borrado de filas concretas

Al utilizar la cláusula WHERE con DELETE se pueden especificar ciertas filas para borrarlas de la tabla. Por ejemplo, para borrar todas las filas de la tabla *items* que tengan un valor de *toys* en *item_category* hay que ejecutar la instrucción siguiente:

```
DELETE FROM items
WHERE item_category ='toys'
GO
```

Esta instrucción borra una fila de la tabla items.

Se puede emplear una segunda cláusula FROM con uno o más orígenes de tabla para especificar otras tablas y vistas que pueden utilizarse en la condición de búsqueda de WHERE. Por ejemplo, para borrar filas de la tabla *items* que se correspondan con las filas de la tabla *two newest items*, hay que ejecutar la instrucción siguiente:

```
DELETE    items
FROM      two_newest_items
WHERE     items.item_id=two_newest_items.item_id
GO
```

Hay que tener en cuenta que en esta instrucción se ha omitido la primera palabra clave opcional FROM. Las dos filas de la tabla *two_newest_items* tienen valores de 2 y 3 en *item_id* La tabla *items* contiene valores *item_id* de 1 y 2, por lo que la fila con un valor 2 de *item_id* (la que coincidía con la condición de búsqueda) queda borrada. Las dos filas de la tabla *two_newest_items* (el origen de la tabla) no se ven afectadas.

Borrado de todas las filas

Para borrar todas las filas de una tabla hay que emplear DELETE sin instrucciones WHERE. La siguiente instrucción DELETE borra todas las filas de la tabla *two_newest_items* de la página siguiente.

```
DELETE FROM two newest_items
GO
```

La tabla *two_newest_items* es ahora una tabla vacía -no contiene ningún dato. Si se desea borrar también la definición de la tabla hay que utilizar el comando DROP TABLE, como se muestra a continuación.

```
DROP TABLE two_newest_items
GO
```