# Capítulo 11

# Cómo crear y mantener bases de datos, tablas y secuencias con sentencias SQL

# Objetivos

## Aplicado

- Dado un diseño de base de datos completo, escriba las instrucciones DDL de SQL para crear la base de datos, incluidas todas las tablas, relaciones, restricciones, índices y secuencias.

## Conocimiento

- Describa cómo cada uno de estos tipos de restricciones restringe los valores que se pueden almacenar en una tabla: NOT NULL, PRIMARY KEY, UNIQUE, CHECK y FOREIGN KEY (o REFERENCES).
- Describa la diferencia entre una restricción de nivel de columna y una restricción de nivel de tabla.
- Explicar en qué se diferencian las opciones CASCADE y NO ACTION en la aplicación de la integridad referencial en las eliminaciones y actualizaciones.

## Objetivos (cont.)

- Describa el uso de una secuencia.
- Describir el uso de un script que contiene uno o varios lotes para crear una base de datos.

# Instrucciones DDL para crear, modificar y eliminar objetos

- CREATE DATABASE
- CREATE TABLE
- CREATE INDEX
- CREATE SEQUENCE
- CREATE FUNCTION
- CREATE PROCEDURE
- CREATE TRIGGER
- CREATE VIEW
- ALTER TABLE
- ALTER SEQUENCE
- ALTER FUNCTION

# Instrucciones DDL para crear, modificar y eliminar objetos (continuación)

ALTER PROCEDURE

ALTER TRIGGER

ALTER VIEW

DROP DATABASE

DROP TABLE

DROP SEQUENCE

DROP INDEX

DROP FUNCTION

DROP PROCEDURE

DROP TRIGGER

DROP VIEW

## Reglas de formato para identificadores

- El primer carácter de un identificador debe ser una letra, tal como se define en el estándar Unicode 2.0, un carácter de subrayado (_), un signo de arroba (@) o un signo de número (#).
- Todos los caracteres después del primero deben ser una letra, tal como se define en el estándar Unicode 2.0, un número, un signo arroba, un signo de dólar ($), un signo numérico o un guión bajo.
- Un identificador no puede ser una palabra clave reservada de Transact-SQL.
- Un identificador no puede contener espacios ni caracteres especiales distintos de los ya mencionados.

## Valid regular identifiers

```
Employees
#PaidInvoices
ABC$123
Invoice_Line_Items
@TotalDue
```

## Valid delimited identifiers

```
[%Increase]
"Invoice Line Items"
[@TotalDue]
```

# Basic syntax of the CREATE DATABASE statement

```
CREATE DATABASE database_name
    [ON [PRIMARY] (FILENAME = 'file_name')]
    [FOR ATTACH]
```

# Create a new database

```
CREATE DATABASE New_AP;
```

## The response from the system

```
Command(s) completed successfully.
```

# Attach an existing database file

```
CREATE DATABASE Test_AP
    ON PRIMARY (FILENAME =
      'C:\Murach\SQL Server 2012\Databases\Test_AP.mdf')
    FOR ATTACH;
```

## The response from the system

```
Command(s) completed successfully.
```

# Basic syntax of the CREATE TABLE statement

```
CREATE TABLE table_name
(column_name_1 data_type [column_attributes]
[, column_name_2 data_type [column_attributes]]...
[, table_attributes])
```

# Common column attributes

- `NULL|NOT NULL`
- `PRIMARY KEY|UNIQUE`
- `IDENTITY`
- `DEFAULT default_value`
- `SPARSE`

## Create a table without column attributes

```
CREATE TABLE Vendors
(VendorID       INT,
VendorName      VARCHAR(50));
```

## Create a table with column attributes

```
CREATE TABLE Invoices
(InvoiceID      INT             PRIMARY KEY IDENTITY,
VendorID        INT             NOT NULL,
InvoiceDate     SMALLDATETIME   NULL,
InvoiceTotal    MONEY           NULL DEFAULT 0);
```

## A column definition that uses the SPARSE attribute

```
VendorAddress2  VARCHAR(50) SPARSE NULL
```

## Basic syntax of the CREATE INDEX statement

```
CREATE [CLUSTERED|NONCLUSTERED] INDEX index_name
    ON table_name (col_name_1 [ASC|DESC]
                  [, col_name_2 [ASC|DESC]]...)
 [WHERE filter-condition]
```

## Create a nonclustered index on a single column

```
CREATE INDEX IX_VendorID
    ON Invoices (VendorID);
```

## Create a nonclustered index on two columns

```
CREATE INDEX IX_Invoices
    ON Invoices (InvoiceDate DESC, InvoiceTotal);
```

## Note

- SQL Server automatically creates a clustered index for a table's primary key.

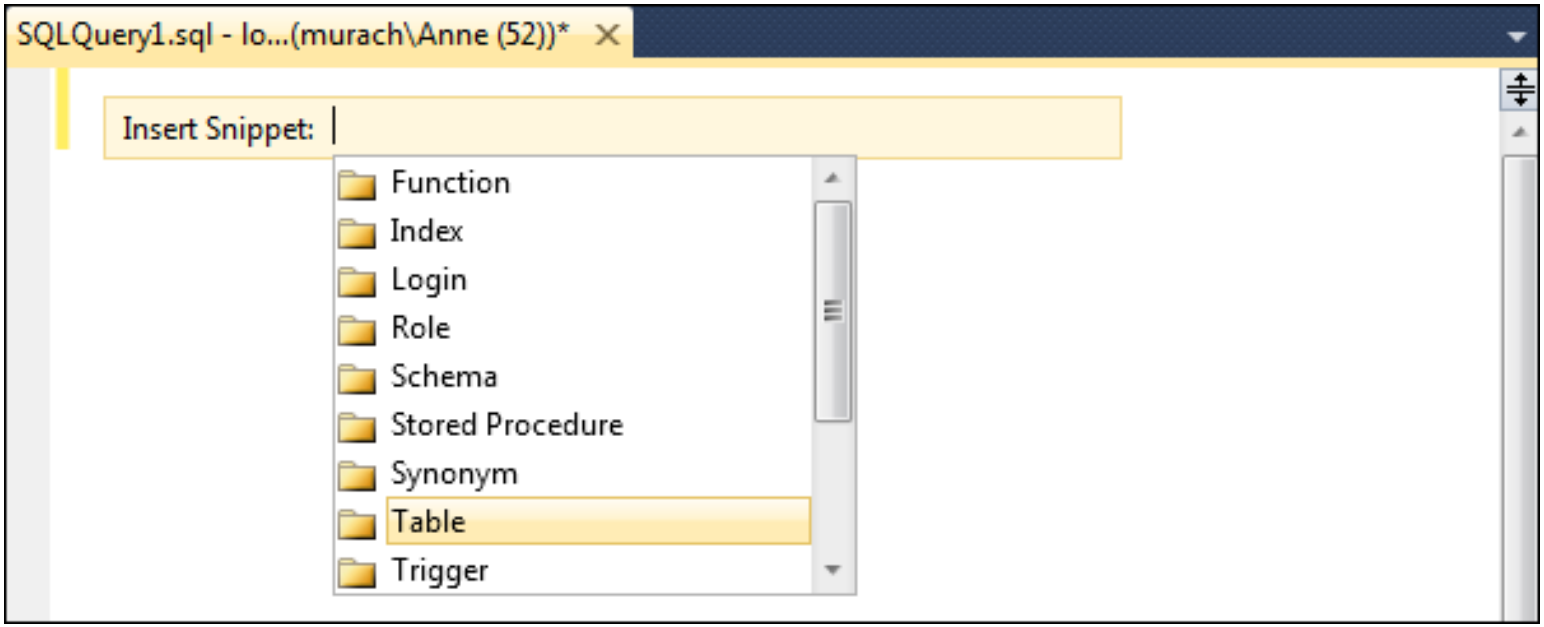
## Create a filtered index for a subset of data in a column

```
CREATE INDEX IX_InvoicesPaymentFilter
    ON Invoices (InvoiceDate DESC, InvoiceTotal)
WHERE PaymentDate IS NULL;
```

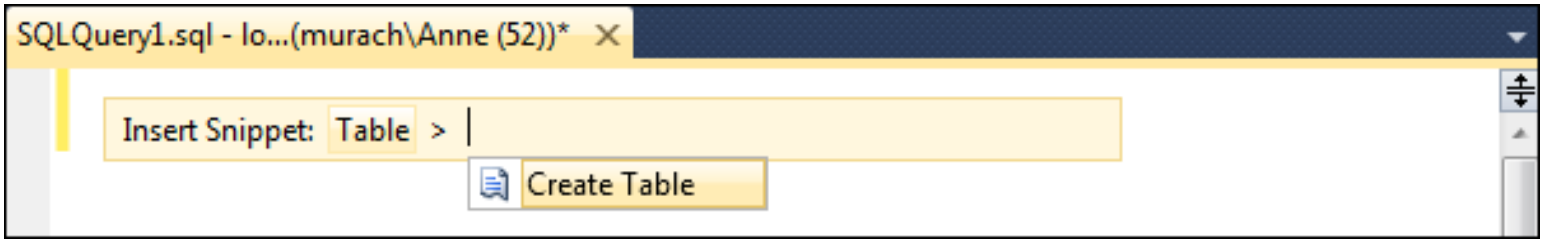
## Create a filtered index for categories in a column

```
CREATE INDEX IX_InvoicesDateFilter
    ON Invoices (InvoiceDate DESC, InvoiceTotal)
WHERE InvoiceDate > '2012-02-01';
```

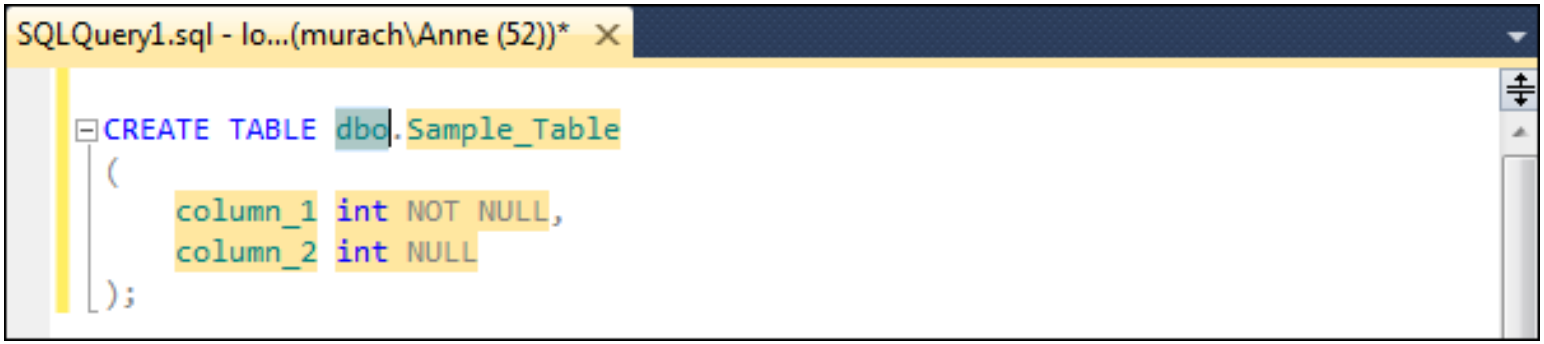
# The snippet picker with a list of object folders



# The snippet picker with the list of snippets for a table




# The CREATE TABLE snippet after it has been inserted



# Terms

- Archivo de registro de transacciones
- Adjuntar un archivo de base de datos
- Índice de tabla completa
- Índice filtrado

# Column-level constraints

| Constraint | Description |
|---|---|
| `NOT NULL` | Prevents null values from being stored in the column. |
| `PRIMARY KEY` | Requires that each row in the table have a unique value in the column. Null values are not allowed. |
| `UNIQUE` | Requires that each row in the table have a unique value in the column. |
| `CHECK` | Limits the values for a column. |
| `[FOREIGN KEY] REFERENCES` | Enforces referential integrity between a column in the new table and a column in a related table. |

# Table-level constraints

| Constraint | Description |
|---|---|
| `PRIMARY KEY` | Requires that each row in the table have a unique set of values over one or more columns. Null values are not allowed. |
| `UNIQUE` | Requires that each row in the table have a unique set of values over one or more columns. |
| `CHECK` | Limits the values for one or more columns. |
| `[FOREIGN KEY] REFERENCES` | Enforces referential integrity between one or more columns in the new table and one or more columns in the related table. |

# Create a table with a two-column primary key constraint

```
CREATE TABLE InvoiceLineItems1
(InvoiceID                  INT          NOT NULL,
InvoiceSequence             SMALLINT     NOT NULL,
InvoiceLineItemAmount       MONEY        NOT NULL,
PRIMARY KEY (InvoiceID, InvoiceSequence));
```

## Create a table with two column-level check constraints

```
CREATE TABLE Invoices1
(InvoiceID        INT   NOT NULL IDENTITY PRIMARY KEY,
InvoiceTotal      MONEY NOT NULL
                  CHECK (InvoiceTotal >= 0),
PaymentTotal      MONEY NOT NULL DEFAULT 0
                  CHECK (PaymentTotal >= 0));
```

## The same check constraints coded at the table level

```
CREATE TABLE Invoices2
(InvoiceID        INT   NOT NULL IDENTITY PRIMARY KEY,
InvoiceTotal      MONEY NOT NULL,
PaymentTotal      MONEY NOT NULL DEFAULT 0,
CHECK ((InvoiceTotal >= 0) AND (PaymentTotal >= 0)));
```

## The syntax of a check constraint

```
CHECK (condition)
```

## A column-level check constraint

```
CREATE TABLE Invoices3
(InvoiceID        INT   NOT NULL IDENTITY PRIMARY KEY,
InvoiceTotal      MONEY NOT NULL CHECK (InvoiceTotal > 0));
```

## An INSERT statement that fails due to the check constraint

```
INSERT Invoices3
VALUES (-100);
```

## The response from the system

```
The INSERT statement conflicted with the CHECK constraint
"CK__Invoices3__Invoi__0BC6C43E". The conflict occurred in
database "New_AP", table "dbo.Invoices3", column
'InvoiceTotal'.
The statement has been terminated.
```

## A table-level check constraint

```
CREATE TABLE Vendors1
(VendorCode       CHAR(6)      NOT NULL PRIMARY KEY,
VendorName        VARCHAR(50)  NOT NULL,
CHECK ((VendorCode LIKE '[A-Z][A-Z][0-9][0-9][0-9][0-9]')
  AND  (LEFT(VendorCode,2) = LEFT(VendorName,2))));
```

## An INSERT statement that fails due to the check constraint

```
INSERT Vendors1
VALUES ('Mc4559','Castle Printers, Inc.');
```

## The response from the system

```
The INSERT statement conflicted with the CHECK constraint
"CK__Vendors1__164452B1". The conflict occurred in database
"New_AP", table "dbo.Vendors1".
The statement has been terminated.
```

# The syntax of a column-level foreign key constraint

```
[FOREIGN KEY] REFERENCES ref_table_name (ref_column_name)
    [ON DELETE {CASCADE|NO ACTION}]
    [ON UPDATE {CASCADE|NO ACTION}]
```

# The syntax of a table-level foreign key constraint

```
FOREIGN KEY (column_name_1 [, column_name_2]...)
    REFERENCES ref_table_name (ref_column_name_1
                               [, ref_column_name_2]...)
    [ON DELETE {CASCADE|NO ACTION}]
    [ON UPDATE {CASCADE|NO ACTION}]
```

# A column-level foreign key constraint

## A statement that creates the primary key table

```
CREATE TABLE Vendors9
(VendorID          INT NOT NULL PRIMARY KEY,
VendorName         VARCHAR(50) NOT NULL);
```

## A statement that creates the foreign key table

```
CREATE TABLE Invoices9
(InvoiceID         INT NOT NULL PRIMARY KEY,
VendorID           INT NOT NULL REFERENCES Vendors9
(VendorID),
InvoiceTotal       MONEY NULL);
```

# A column-level foreign key constraint (continued)

## An INSERT statement that fails because a related row doesn't exist

```
INSERT Invoices9
VALUES (1, 99, 100);
```

## The response from the system

```
The INSERT statement conflicted with the FOREIGN KEY
constraint "FK__Invoices9__Vendo__1367E606". The conflict
occurred in database "New_AP", table "dbo.Vendors9",
column 'VendorID'.
The statement has been terminated.
```

# Terms

- Constraint
- Column-level constraint
- Table-level constraint
- Check constraint
- Foreign key constraint
- Reference constraint
- Cascading delete
- Cascading update

## The syntax of the DROP INDEX statement

```
DROP INDEX index_name_1 ON table_name_1
        [, index_name_2 ON table_name_2]...
```

## Delete an index from the Invoices table

```
DROP INDEX IX_Invoices ON Invoices;
```

## Note

- You can't delete an index that's based on a primary key or unique key constraint. To do that, you have to use the ALTER TABLE statement.

## The syntax of the DROP TABLE statement

```
DROP TABLE table_name_1 [, table_name_2]...
```

## Delete a table from the current database

```
DROP TABLE Vendors1;
```

## Qualify the table to be deleted

```
DROP TABLE New_AP.dbo.Vendors1;
```

## Notes

- You can't delete a table if a foreign key constraint in another table refers to that table.
- When you delete a table, all of the data, indexes, triggers, and constraints are deleted. Any views or stored procedures associated with the table must be deleted explicitly.

## The syntax of the DROP DATABASE statement

```
DROP DATABASE database_name_1 [, database_name_2]...
```

## A statement that deletes a database

```
DROP DATABASE New_AP;
```

# The basic syntax of the ALTER TABLE statement

```
ALTER TABLE table_name [WITH CHECK|WITH NOCHECK]
{ADD new_column_name data_type [column_attributes] |
 DROP COLUMN column_name |
 ALTER COLUMN column_name new_data_type [NULL|NOT NULL] |
 ADD [CONSTRAINT] new_constraint_definition |
 DROP [CONSTRAINT] constraint_name}
```

## Add a new column

```
ALTER TABLE Vendors
ADD LastTranDate SMALLDATETIME NULL;
```

## Drop a column

```
ALTER TABLE Vendors
DROP COLUMN LastTranDate;
```

## Add a new check constraint

```
ALTER TABLE Invoices WITH NOCHECK
ADD CHECK (InvoiceTotal >= 1);
```

## Add a foreign key constraint

```
ALTER TABLE InvoiceLineItems WITH CHECK
ADD FOREIGN KEY (AccountNo) REFERENCES
GLAccounts(AccountNo);
```

## Change the data type of a column

```
ALTER TABLE InvoiceLineItems
ALTER COLUMN InvoiceLineItemDescription VARCHAR(200);
```

# The syntax of the CREATE SEQUENCE statement

```
CREATE SEQUENCE sequence_name
    [AS integer_type]
    [START WITH starting_integer]
    [INCREMENT BY increment_integer]
    [{MINVALUE minimum_integer | NO MINVALUE}]
    [{MAXVALUE maximum_integer | NO MAXVALUE}]
    [{CYCLE|NOCYCLE}]
    [{CACHE cache_size|NOCACHE}]
```

## Create a sequence that starts with 1

```
CREATE SEQUENCE TestSequence1
    START WITH 1;
```

## Specify a starting value and an increment

```
CREATE SEQUENCE TestSequence2
    START WITH 10
    INCREMENT BY 10;
```

## Specify all optional parameters

```
CREATE SEQUENCE TestSequence3
    AS int
    START WITH 100 INCREMENT BY 10
    MINVALUE 0 MAXVALUE 1000000
    CYCLE CACHE 10;
```

## Create a table with a sequence column

```
CREATE TABLE SequenceTable(
    SequenceNo      INT,
    Description     VARCHAR(50));
```

## Insert the next value for a sequence

```
INSERT INTO SequenceTable
VALUES (NEXT VALUE FOR TestSequence3, 'First inserted row')
INSERT INTO SequenceTable
VALUES (NEXT VALUE FOR TestSequence3,
        'Second inserted row');
```

## Get the current value of the sequence

```
SELECT current_value FROM sys.sequences
WHERE name = 'TestSequence3';
```

| | current_value |
|---|---|
| 1 | 110 |

## The syntax of the DROP SEQUENCE statement

```
DROP SEQUENCE sequence_name1[, sequence_name2]...
```

## A statement that drops a sequence

```
DROP SEQUENCE TestSequence2;
```

# The syntax of the ALTER SEQUENCE statement

```
ALTER SEQUENCE sequence_name
    [RESTART [WITH starting_integer]]
    [INCREMENT BY increment_integer]
    [{MINVALUE minimum_integer | NO MINVALUE}]
    [{MAXVALUE maximum_integer | NO MAXVALUE}]
    [{CYCLE|NOCYCLE}]
    [{CACHE cache_size|NOCACHE}]
```

# A statement that alters a sequence

```
ALTER SEQUENCE TestSequence1
    INCREMENT BY 9
    MINVALUE 1 MAXVALUE 999999
    CACHE 9
    CYCLE;
```

# The SQL script that creates the AP database

```
CREATE DATABASE AP;
GO

USE AP;
CREATE TABLE Terms
(TermsID                INT             NOT NULL PRIMARY KEY,
TermsDescription        VARCHAR(50)     NOT NULL,
TermsDueDays            SMALLINT        NOT NULL);

CREATE TABLE GLAccounts
(AccountNo              INT             NOT NULL PRIMARY KEY,
AccountDescription      VARCHAR(50)     NOT NULL);
```

# The SQL script (cont.)

```
CREATE TABLE Vendors
(VendorID                INT             NOT NULL IDENTITY
PRIMARY KEY,
VendorName               VARCHAR(50)     NOT NULL,
VendorAddress1           VARCHAR(50)     NULL,
VendorAddress2           VARCHAR(50)     SPARSE NULL,
VendorCity               VARCHAR(50)     NOT NULL,
VendorState              CHAR(2)         NOT NULL,
VendorZipCode            VARCHAR(20)     NOT NULL,
VendorPhone              VARCHAR(50)     NULL,
VendorContactLName       VARCHAR(50)     NULL,
VendorContactFName       VARCHAR(50)     NULL,
DefaultTermsID           INT             NOT NULL
                         REFERENCES Terms(TermsID),
DefaultAccountNo         INT             NOT NULL
                         REFERENCES GLAccounts(AccountNo));
```

# The SQL script (cont.)

```
CREATE TABLE Invoices
(InvoiceID              INT           NOT NULL IDENTITY
PRIMARY KEY,
VendorID                INT           NOT NULL
                        REFERENCES Vendors(VendorID),
InvoiceNumber           VARCHAR(50)   NOT NULL,
InvoiceDate             SMALLDATETIME NOT NULL,
InvoiceTotal            MONEY         NOT NULL,
PaymentTotal            MONEY         NOT NULL DEFAULT 0,
CreditTotal             MONEY         NOT NULL DEFAULT 0,
TermsID                 INT           NOT NULL
                        REFERENCES Terms(TermsID),
InvoiceDueDate          SMALLDATETIME NOT NULL,
PaymentDate             SMALLDATETIME NULL);
```

# The SQL script (cont.)

```
CREATE TABLE InvoiceLineItems
(InvoiceID                  INT            NOT NULL
                            REFERENCES Invoices(InvoiceID),
InvoiceSequence             SMALLINT       NOT NULL,
AccountNo                   INT            NOT NULL
                            REFERENCES GLAccounts(AccountNo),
InvoiceLineItemAmount       MONEY          NOT NULL,
InvoiceLineItemDescription VARCHAR(100) NOT NULL,
PRIMARY KEY (InvoiceID, InvoiceSequence));
```

# The SQL script (cont.)

```
CREATE INDEX IX_Invoices_VendorID
    ON Invoices (VendorID);
CREATE INDEX IX_Invoices_TermsID
    ON Invoices (TermsID);
CREATE INDEX IX_Vendors_TermsID
    ON Vendors (DefaultTermsID);
CREATE INDEX IX_Vendors_AccountNo
    ON Vendors (DefaultAccountNo);
CREATE INDEX IX_InvoiceLineItems_AccountNo
    ON InvoiceLineItems (AccountNo);
CREATE INDEX IX_VendorName
    ON Vendors (VendorName);
CREATE INDEX IX_InvoiceDate
    ON Invoices (InvoiceDate DESC);
```

# Terms

- Script
- Batch