¿Qué es el T-SQL (Transact SQL)?

El T-SQL es un lenguaje relacional orientado a datos, que se preocupa más por definir el conjunto de resultados que se pretende obtener que por los aspectos físicos de recolección. No confunda a T-SQL con un lenguaje de programación, ya que ambos tienen alcances distintos.

T-SQL provee de dos conjuntos de sentencias y estructuras que pueden dividirse en DDL (Data Definition Language, Lenguaje de definición de datos) y DML (Data Manipulation Language, Lenguaje de Manipulación de Datos).

Los comandos DDL (Data Definition Language) se utilizan para manipular objetos en el ámbito de una base de datos y se basan, generalmente, en sentencias CREATE, ALTER y DROP.

Los comandos DML (Data Manipulation Language) conforman el conjunto de instrucciones necesario para manipular datos en T-SQL. Algunos ejemplos son SELECT, INSERT, UPDATE, DELETE, etc.

CREATE

CREATE es una sentencia DDL (Data Definition Language) que se utiliza para crear objetos en el lenguaje SQL.

Create database

Para crear una base de datos que contenga las nuevas tablas, escriba el código siguiente en una ventana de consultas.

```
USE master;
GO

-- Creación de una nueva base de datos llamada XXX.
CREATE DATABASE XXX
GO
```

Presione F5 para ejecutar la consulta y crear la base de datos.

Cambie la conexión desde la ventana de consultas a la base de datos XXX.

En una ventana de consultas, escriba y ejecute el siguiente código para cambiar la conexión a la base de datos XXX.

```
USE XXX
GO
```

Create Table

Para crear una tabla, debe proporcionar un nombre para ésta además de los nombres y los tipos de datos de cada columna de la tabla. También es recomendable indicar si se permiten valores NULL en cada columna.

Todas las tablas tienen una clave principal, que se compone de una o varias columnas de la tabla. Una clave principal siempre es única.

Para crear una tabla

En una consulta escriba y ejecute el siguiente código para crear una tabla sencilla denominada Productos. Las columnas de la tabla son ProductoID, NombreProducto, PrecioProducto y DescripcionProducto. Int, varchar(25), money y text son todos los tipos de datos. Solo las columnas PrecioProducto y DescripcionProducto pueden no tener datos cuando se inserta o cambia una fila.

```
CREATE TABLE Productos
 (
  ProductoID char(4) NOT NULL,
  NombreProducto varchar(25) NOT NULL,
  PrecioProducto money NULL,
  DescripcionProducto text NULL,
  CONSTRAINT PK_Productos PRIMARY KEY (ProductoID)
)
GO
```

Para crear una tabla con dos o más campos claves

Para crear la misma tabla que ya describimos pero agregándole el campo ProveedorID que será clave primaria junto a ProductoID, formando juntos una clave compuesta o concatenada, deberemos escribir en una consulta.

```
CREATE TABLE Productos
 (
  ProveedorID_Prod char(4) NOT NULL,
  ProductoID_Prod char(4) NOT NULL,
  NombreProducto_Prod varchar(25) NOT NULL,
```

```
    PrecioProducto_Prod money NULL,
    DescripcionProducto_Prod text NULL,
    Constraint PK_Productos PRIMARY KEY (ProveedorID_Prod, ProductoID_Prod)
)
GO
```

¿Cómo eliminar una Tabla?

Administrador Corporativo: accediendo a la sección Tablas, seleccionando la misma y pulsando la tecla suprimir o el botón secundario del Mouse y elegir eliminar.

Ventana de Consultas: Ejecutando la siguiente instrucción

```
DROP TABLE NOMBRE_DE_LA_TABLA
```

La sentencia DROP elimina cualquier elemento de la base de datos, hasta la misma base de datos si es que lo requerimos. Algunos Ejemplos a tener en cuenta

DROP TABLE Elimina una tabla

DROP VIEW Elimina una vista determinada

DROP PROCEDURE Elimina un procedimiento almacenado

DROP DATABASE Elimina una Base de datos

DROP FUNCTION Elimina una función de la base de datos

Procedimientos almacenados

Consisten en rutinas almacenadas bajo un nombre en el servidor que pueden devolver resultados tabulares y mensajes al cliente que los utiliza, invocar sentencias de manipulación y definición de datos y retomar parámetros de salida.

Cuando se reciben sentencias T-SQL desde una aplicación cliente, el servidor ejecuta las siguientes acciones:

- Analiza el comando y crea el área de secuencia de ejecución: el servidor verifica que la sintaxis del comando sea correcta y transforma en el lenguaje interno con el que trabaja. A este formato interno se lo conoce como árbol de ejecución.

- Compila el árbol generando el plan de ejecución de la consulta verificando cuál es el plan óptimo, la seguridad y los permisos de los objetos, las restricciones, etc.

- Ejecuta la consulta invocando al responsable de cada una de ellas.

Estos pasos se ejecutan para cada consulta enviada al servidor. La gran ventaja de utilizar procedimientos almacenados consiste en que su plan de ejecución se almacena en la caché del servidor, con lo cual se obtiene su desempeño muy elevado del motor de SQL Server si, las consultas más comunes que utiliza cualquier aplicación, se ejecutan como procedimiento almacenado.

La sentencia Create Procedure permite crear un procedimiento almacenado.

```
CREATE PROCEDURE NOMBRE
   @PARAMETRO TIPODATO, @PARAMETRO TIPODATO
   AS
   CONSULTA
GO
```

Forma de ejecutar un procedimiento: en la ventana de consultas escriba

```
EXECUTE NOMBREPROCEDIMIENTO
ó
EXEC NOMBREPROCEDIMIENTO
```

Por Ejemplo,

```
CREATE PROCEDURE SP_TRAERPRODUCTO
      @CODIGO CHAR(4)
      AS
      SELECT DESCRIPCION, MARCA, COSTO FROM PRODUCTOS
      WHERE CODIGO=@CODIGO
GO
```

Forma de ejecutar el procedimiento: en la ventana de consultas escriba

```
EXECUTE SP_TRAERPRODUCTO 'CR01'

ó

EXEC SP_TRAERPRODUCTO 'CR02'
```

… donde el número 2 es el valor asignado a la variable @CODIGO. En caso de que el tipo de dato de la variable sea de cadena de caracteres deberá escribirlo entre 'comillas simples'.

CREATE VIEW (Vistas)

Crea una tabla virtual cuyo contenido (columnas y filas) se define mediante una consulta. Utilice esta instrucción para crear una vista de los datos de una o varias tablas de la base de datos. Por ejemplo, una vista se puede utilizar para lo siguiente:

- Para centrar, simplificar y personalizar la percepción de la base de datos para cada usuario.
- Como mecanismo de seguridad, que permite a los usuarios obtener acceso a los datos por medio de la vista, pero no les conceden el permiso de obtener acceso directo a las tablas base subyacentes de la vista.
- Para proporcionar una interfaz compatible con versiones anteriores para emular una tabla cuyo esquema ha cambiado.

El código que se encuentra a continuación permite crear una vista en SQL Server:

```
CREATE VIEW NOMBRE_DE_LA_VISTA

  AS

 SELECT CODIGO, DESCRIPCION, VALORVENTA FROM PRODUCTOS
```

En donde el AS permite saber donde comienza a ejecutarse la instrucción de consulta.

Para ver el contenido de una vista basta con realizar la siguiente instrucción:

```
SELECT * FROM NOMBRE_DE_LA_VISTA
```

Restricción CHECK

Una columna puede tener cualquier número de restricciones CHECK y la condición puede incluir varias expresiones lógicas combinadas con AND y OR. Varias restricciones CHECK para una columna se validan en el orden en que se crean.

La condición de búsqueda debe dar como resultado una expresión booleana y no puede hacer referencia a otra tabla. Una restricción CHECK en el nivel de columna sólo puede

hacer referencia a la columna restringida, y una restricción CHECK en el nivel de tabla sólo puede hacer referencia a columnas de la misma tabla.

Las restricciones CHECK y las reglas sirven para la misma función de validación de los datos durante las instrucciones INSERT y DELETE. Cuando hay una regla y una o más restricciones CHECK para una columna o columnas, se evalúan todas las restricciones.

En la declaración de una tabla con un solo valor para chequear…

CODCARRERA SMALLINT NOT NULL CHECK(CODCARRERA>=1)

Y con dos valores para el mismo campo…

ESTADO CHAR(2) NOT NULL CHECK (ESTADO='SI' OR ESTADO='NO'), --SI O NO

Restricción DEFAULT

El Uso del DEFAULT nos permite establecer el valor predeterminado en caso de que el usuario no ingrese un valor.

```
...

Campo smallint NOT NULL DEFAULT 1,

...
```

Relaciones entre tablas.

El siguiente ejemplo realiza la referencia con el campo CP de la Tabla Localidades, en donde se asegura que exista la localidad en el momento de insertar el registro en la tabla personas.

En la creación de Tabla se pueden establecer como clave foránea los campos que luego formarán parte de una relación, colocando además restricciones.

```
CREATE TABLE Personas

(

 . . .

 Nombre_PER VARCHAR(30) NOT NULL,

 CP_PER CHAR(10) NOT NULL, . . .

 CONSTRAINT FK_Personas_Localidades FOREIGN KEY (CP_PER) REFERENCES
Localidades(CP_LOC),

 . . .

 )
```

Si se quisiera hacer referencia a una clave primaria de dos campos de otra tabla se deberá agregar los campos en el mismo orden que se encuentran en la tabla de referencia.

CONSTRAINT FK_DetFact_Articulos FOREIGN KEY (CodProveedor_df, CodArticulo_df) REFERENCES ArtXProv (CodProveedor_axp, CodArticulo_axp)

…

)

GO

INSERT

El comando INSERT INTO anexa un registro al final de una tabla que contiene los valores de campo especificados.

En este caso creamos una tabla llamada 'pruebas' con tres campos:

```
CREATE TABLE Pruebas
(
CodPrueba_Prue INT IDENTITY(1,1) NOT NULL,
NombPrueba_Prue VARCHAR(20) NOT NULL,
CantPrueba_Prue INT NOT NULL,
CONSTRAINT PK_Pruebas PRIMARY KEY (CodPrueba_Prue)
)
```

La forma básica de insertar un registro es la siguiente…

```
INSERT INTO pruebas (NombPrueba_Prue, CantPrueba_Prue)
VALUES('casa',10)
```

…y de esta forma, si quisiera insertar múltiples registros sería así.

```
INSERT INTO pruebas (NombPrueba_Prue, CantPrueba_Prue)
VALUES('casa',10)
INSERT INTO pruebas (NombPrueba_Prue, CantPrueba_Prue)
VALUES('coco',8)
```

Las últimas versiones de SQL Server aceptan esta otra forma de inserción múltiple, pero nosotros no la utilizaremos porque las versiones más antiguas no la aceptan

```
INSERT INTO pruebas (NombPrueba_Prue, CantPrueba_Prue)
VALUES('casa',10),('coco',8)
```

…y en cambio utilizaremos esta que sí es aceptada por las versiones anteriores

```
INSERT INTO pruebas (NombPrueba_Prue, CantPrueba_Prue)
SELECT 'casa',10 UNION
SELECT 'coco',8
```

Una variante más aceptada con este último método SELECT UNION…

Creamos la tabla ‘desde’ (nótese que el campo cod_desde no es autonumérico)

```
CREATE TABLE desde
(
cod_desde int primary key,
Nomb_desde varchar(20) not null,
cant_desde int
)
```

Insertamos dos registros con el método SELECT UNION

```
INSERT INTO desde (cod_desde, Nomb_desde, cant_desde)
SELECT 10, 'auto',10 UNION
SELECT 11, 'avión',8
```

Insertamos los registros de la tabla ‘desde’ en la tabla ‘pruebas’ con una subconsulta.

```
INSERT INTO pruebas (NombPrueba_Prue, CantPrueba_Prue)
SELECT Nomb_desde, cant_desde from desde
```