

Capítulo 8

Gestionando los datos con lenguaje SQL

Al finalizar el capítulo, el alumno podrá:

- Elaborar sentencias de actualización de datos.

Temas:

1. Sentencia INSERT
2. Sentencia UPDATE
3. Sentencia DELETE


1. Sentencia INSERT

Sentencia INSERT

- Se utiliza para añadir registros a una tabla.
- El nombre de la tabla es utilizado para indicar el destino.
- El nombre de las columnas a donde se irán los valores.

INSERT INTO NOMBRE TABLA
(COLUMNAS)
VALUES (VALORES)

INSERT INTO TB_DISTRITO (COD_DIS,
NOM_DIS, COD_VEN) VALUES ('099',
'MAGDALENA', 'V01')

B - 4Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.

1.1 Definición

La sentencia de INSERT se utiliza para añadir registros a las tablas de la base de datos. Además, implementa la típica operación de “Alta” de registros.

1.2 Parámetros del comando INSERT

El formato de la sentencia es:

```
INSERT INTO Nombre_tabla [(nombre_columna,...)] VALUES (expr,...)
```

Nombre_Tabla es el nombre de la tabla donde se desea ingresar los nuevos datos.

Nombre_Columna es una lista opcional de nombres de campo en los que se insertarán valores en el mismo número y orden que se especificarán en la cláusula VALUES. Si no se especifica la lista de campos, los valores de **expr** en la cláusula VALUES deben ser tantos, como campos tenga la tabla y en el mismo orden que se definieron al crear la tabla.

Expr es una lista de expresiones o valores constantes, separados por comas, para dar valor a los distintos campos del registro que se añadirán a la tabla. Las cadenas de caracteres deberán estar encerradas entre comillas.

A continuación, presentamos algunos ejemplos que demuestran la funcionalidad básica de la instrucción INSERT usando la sintaxis mínima requerida:

a. Insertar una sola fila de datos

En el siguiente ejemplo se inserta una fila en la tabla **Shippers** en la base de datos de Northwind. Las columnas de esta tabla son **ShipperID**, **CompanyName**, **Phone**. Dado que los valores de todas las columnas se suministran e incluyen en el mismo orden que las columnas de la tabla, no es necesario especificar los nombres de columna en la lista de columnas.

```
INSERT INTO Shippers
VALUES (4, 'Bongo International', '(503) 555-9930')
```

b. Insertar varias filas de datos

En el ejemplo siguiente se usa el constructor de valores de tablas para insertar tres filas en la tabla **Shippers** de la base de datos Northwind en una sola instrucción **INSERT**. Dado que los valores para todas las columnas se suministran e incluyen en el mismo orden que las columnas de la tabla, no es necesario especificar los nombres de columna de la lista de columnas.

```
INSERT INTO Shippers
VALUES (5, 'Shipito', '(503) 555-9935'),
      (6, 'comGateway', '(503) 555-9936'),
      (7, 'BorderLinux', '(503) 555-9937')
```

c. Insertar datos que no están en el mismo orden que las columnas de la tabla

En el siguiente ejemplo se utiliza una lista de columnas para especificar de forma explícita los valores insertados en cada columna. El orden de las columnas de la tabla **Shippers** en la base de datos Northwind es **CompanyName**, **ShipperID**, **Phone**; no obstante, las columnas no se incluyen en dicho orden en **nombre_columna**.

```
INSERT INTO Shippers(CompanyName, ShipperID, Phone)
VALUES ('Fishisfast global forwarding', 8, '(503) 555-9945');
```

d. Insertar datos en forma masiva

El ejemplo siguiente crea una tabla e inserta los datos en ella desde otra tabla.

```
CREATE TABLE SuppliersHistory
( SupplierID    int NOT NULL,
  CompanyName  varchar(40) NOT NULL,
  ContactName  varchar(30) NULL,
  ContactTitle  varchar(30) NULL,
  Address      varchar(60) NULL,
  City         varchar(15) NULL,
  Region      varchar(15) NULL,
  PostalCode   varchar(10) NULL,
  Country      varchar(15) NULL,
  Phone       varchar(24) NULL,
  Fax         varchar(24) NULL,
  HomePage    varchar(max) NULL
)

INSERT INTO SuppliersHistory
(SupplierID, CompanyName, ContactName, ContactTitle, Address, City,
Region, PostalCode, Country, Phone, Fax, HomePage)
SELECT * FROM Suppliers
```

Como se puede observar es posible combinar el comando INSERT con las consultas de selección, para agregar datos específicos sin tener la necesidad de realizarlo uno por uno. No obstante, hay que recordar que la nueva tabla ya debe estar creada y la estructura debe ser compatible.

e. Crear una tabla especificando columnas de varios orígenes

En el ejemplo siguiente se crea la tabla **BackupProducts** en la base de datos Northwind seleccionando siete columnas de varias tablas relacionadas con los productos, proveedores y categorías.

```
SELECT P.ProductID, P.ProductName, S.CompanyName, C.CategoryName,  
       P.QuantityPerUnit, P.UnitPrice, P.UnitsInStock  
INTO BackupProducts  
FROM Products P  
INNER JOIN Suppliers S ON S.SupplierID = P.SupplierID  
INNER JOIN Categories C ON C.CategoryID = P.CategoryID
```

f. Insertar imágenes en un campo Image

Al crear una tabla, es necesario con regularidad almacenar las imágenes de los registros, por ejemplo, la imagen de un empleado, la imagen de un producto, la imagen de una intervención quirúrgica, las fotos de un auto siniestrado en un sistema de seguros vehiculares, etc.

Las imágenes pueden ocupar un poco más de espacio en disco que los tipos texto o los datos numéricos, así como las fechas, si van a ser muchos registros, es recomendable separar las imágenes en un grupo de archivos en donde están los datos del mismo registro usando una partición vertical de la tabla.

En este ejemplo, se va a insertar una fila en la tabla **Categories** de la base de datos **Northwinds**, en esta tabla existe una columna **Picture** con el tipo de dato **Image**; para que funcione correctamente, antes de ejecutar el script vamos a crear la carpeta D:\CategoriesPhotos y colocamos las fotos que deseamos insertar.

```
INSERT INTO Categories (CategoryName, Description, Picture)  
VALUES ('AnotherCategory', 'Another Category',  
       (SELECT * FROM OPENROWSET  
        (BULK 'D:\CategoriesPhotos\AnotherCategory.jpg',  
         SINGLE_BLOB) as CategoryImage))
```

g. Insertar información en un campo XML

Los tipos de datos XML permiten guardar información estructurada que dependa del mismo registro. El uso de los tipos de datos XML es muy efectivo para elementos dependientes que no tengan muchos datos.

En este ejemplo, el tipo de campo XML almacenará los datos de los dependientes de un asegurado.

La table **Asegurados** tiene un campo **Dependientes** de tipo XML llamado AseguradosDependientes.

```
CREATE TABLE Asegurados  
( AseguradosCodigo          char(6),  
  AseguradosNombreCompleto  varchar(200),  
  AseguradosDependientes    XML,  
  AseguradosFechaNacimiento Date,  
  CONSTRAINT AseguradosCodigoPK PRIMARY KEY (Codigo)  
)
```

```

INSERT INTO Asegurados (AseguradosCodigo, AseguradosNombreCompleto,
AseguradosDependientes, AseguradosFechaNacimiento)
VALUES ('852369', 'PEDRO CASTRO NIEVES',
'<Dependientes>
<Dependiente>
<Codigo>7852</Codigo>
<Nombre>Vilma</Nombre>
<Paterno>Pereda</Paterno>
<Materno>Plasencia</Materno>
<Parentesco>Esposa</Parentesco>
</Dependiente>
<Dependiente>
<Codigo>9615</Codigo>
<Nombre>Jose</Nombre>
<Paterno>Casto</Paterno>
<Materno>Pereda</Materno>
<Parentesco>Hijo</Parentesco>
</Dependiente>
<Dependiente>
<Codigo>3578</Codigo>
<Nombre>Paola</Nombre>
<Paterno>Casto</Paterno>
<Materno>Pereda</Materno>
<Parentesco>Hija</Parentesco>
</Dependiente>
</Dependientes>',
'15/09/1990')

```

Al listar los registros de la tabla se muestra el campo XML con las etiquetas escritas.

	Codigo	NombreCompleto	Dependientes	FechaNacimiento
1	852369	PEDRO CASTRO NIEVES	<Dependientes><Dependiente><Codigo>7852</Codigo><...>	1990-09-15

Al pulsar clic en el campo XML se muestra el detalle.

```

Dependientes2.xml  SQLQuery1.sql - GP...A\cma1partida (55))*
<Dependientes>
  <Dependiente>
    <Codigo>7852</Codigo>
    <Nombre>Vilma</Nombre>
    <Paterno>Pereda</Paterno>
    <Materno>Plasencia</Materno>
    <Parentesco>Esposa</Parentesco>
  </Dependiente>
  <Dependiente>
    <Codigo>9615</Codigo>
    <Nombre>Jose</Nombre>
    <Paterno>Casto</Paterno>
    <Materno>Pereda</Materno>
    <Parentesco>Hijo</Parentesco>
  </Dependiente>
  <Dependiente>
    <Codigo>3578</Codigo>
    <Nombre>Paola</Nombre>
    <Paterno>Casto</Paterno>
    <Materno>Pereda</Materno>
    <Parentesco>Hija</Parentesco>
  </Dependiente>
</Dependientes>

```


2. Sentencia UPDATE

Sentencia UPDATE

- Se utiliza para actualizar los registros de una tabla.
- El nombre de la tabla es utilizado para indicar el destino.
- El nombre de las columnas es una lista de campos a actualizar de la tabla.
- Puede incluir una condición mediante la cláusula WHERE.

UPDATE NOMBRE TABLA SET
COLUMNA=EXPRESIÓN... CONDICIÓN

UPDATE TB_DISTRITO SET NOM_DIS
='LIMA'
WHERE COD_DIS = 'D99'

8 - 8Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.

2.1 Definición

La sentencia UPDATE se utiliza para cambiar el contenido de los registros de una tabla de la base de datos.

2.2 Parámetros

La sintaxis de esta sentencia es como se muestra.

```
UPDATE Nombre_tabla SET nombre_columna = expr,...  
[WHERE { condición }]
```

Nombre_Tabla es el nombre de la tabla donde se desea ingresar los nuevos datos.

Nombre_columna es el nombre de columna o campo cuyo valor se desea cambiar. En una misma sentencia UPDATE pueden actualizarse varios campos de cada registro de la tabla.

Expr es el nuevo valor que se desea asignar al campo que le precede. La expresión puede ser un valor constante o una subconsulta. Las cadenas de caracteres deberán estar encerradas entre comillas, mientras que las subconsultas entre paréntesis.

La cláusula WHERE que es opcional, sigue el mismo formato que la vista en la sentencia SELECT y determina qué registros se modificarán.

A continuación, presentamos algunos ejemplos que demuestran la funcionalidad básica de la instrucción UPDATE usando la sintaxis mínima requerida.

a. Usar una instrucción UPDATE simple

En el ejemplo siguiente se actualiza un solo valor de la columna para todos los datos de la tabla **Products**.

```
UPDATE Products
SET UnitPrice = UnitPrice * 1.1
```

b. Actualizar varias columnas

En el siguiente ejemplo se actualizan los valores de las columnas UnitPrice, UnitsInStock y UnitsOnOrder para todas las filas de la tabla **Products**.

```
UPDATE Products
SET UnitPrice = UnitPrice * 1.1,
    UnitsInStock = UnitsInStock + 10,
    UnitsOnOrder = UnitsOnOrder + 5
```

c. Limitar las filas que se actualizan utilizando la cláusula WHERE

En el ejemplo siguiente se utiliza la cláusula **WHERE** para especificar las filas que se van a actualizar. La instrucción actualiza el valor de la columna UnitPrice de la tabla **Products** para todas las filas con un valor existente de '1' en la columna CategoryID.

```
UPDATE Products
SET UnitPrice = UnitPrice * 1.1
WHERE CategoryID = 1
```

d. Especificar un alias de tabla como el objeto de destino

En el siguiente ejemplo se actualizan las filas de la tabla **Products**. El alias de tabla asignado a **Products** de la cláusula **FROM** se especifica como el objeto de destino de la cláusula **UPDATE**.

```
UPDATE P
SET ProductName += ' - (New)'
FROM Products P
INNER JOIN [Order Details] OD ON P.ProductID = OD.ProductID
    AND OD.Quantity > 300
```

e. Usar la instrucción UPDATE con información de otra tabla

En este ejemplo se modifica la columna **UnitPrice** de la tabla **Products** de las órdenes más recientes registradas en la tabla **Orders**.

```
UPDATE Products
SET UnitPrice = P.UnitPrice * 1.10
FROM Products P
INNER JOIN [Order Details] OD ON OD.ProductID = P.ProductID
INNER JOIN Orders O ON O.OrderID = OD.OrderID
                        AND O.OrderDate = (SELECT MAX(OrderDate)
                                           FROM Orders)
```

f. Actualizar una columna Image

En el ejemplo siguiente vamos a actualizar la imagen que se encuentra en la tabla **Categories** en donde el CategoryID sea igual a '9'.

```
UPDATE Categories
SET Picture = (SELECT * FROM OPENROWSET
              (BULK 'D:\CategoriesPhotos\uva.jpg', SINGLE_BLOB) AS T)
WHERE CategoryID = 9
```



3. Sentencia DELETE

Sentencia UPDATE

- Se utiliza para actualizar los registros de una tabla.
- El nombre de la tabla es utilizado para indicar el destino.
- El nombre de las columnas es una lista de campos a actualizar de la tabla.
- Puede incluir una condición mediante la cláusula WHERE.

`UPDATE NOMBRE TABLA SET
COLUMNA=EXPRESIÓN... CONDICIÓN`

`UPDATE TB_DISTRITO SET NOM_DIS
= 'LIMA'
WHERE COD_DIS = 'D99'`

8 - 8Copyright © Todos los Derechos Reservados - Cibertec Perú SAC.

3.1 Definición

Es especialmente útil cuando se desea eliminar varios registros. En una instrucción DELETE con múltiples tablas, se debe incluir el nombre de tabla (Tabla.*). Si especifica más de una tabla para eliminar registros, todas deben tener una relación de muchos a uno. Si se desea eliminar todos los registros de una tabla, eliminar la propia tabla es más eficiente que ejecutar una consulta de borrado.

Asimismo, DELETE puede eliminar registros de una única tabla o desde varios lados de una relación uno a muchos. Las operaciones de eliminación en cascada en una consulta únicamente eliminan desde varios lados de una relación.

Una vez que se han eliminado los registros utilizando una consulta de borrado, no se podrá deshacer la operación. Si se desea saber qué registros se eliminarán, primero se examinan los resultados de una consulta de selección que utilice el mismo criterio y después, se ejecuta la consulta de borrado. Del mismo modo, se deben mantener copias de seguridad de datos. Si se eliminan los registros equivocados podrán ser recuperados desde las copias de seguridad.

3.2 Parámetros

El formato de la sentencia es:

```
DELETE FROM Nombre_Tabla [WHERE { condición }]
```

Nombre_Tabla es el nombre de la tabla donde se desea borrar los datos.

La cláusula **WHERE** que es opcional, sigue el mismo formato que la vista en la sentencia SELECT y determina qué registros se borrarán.

Cada sentencia **DELETE** borra los registros que cumplen la condición impuesta o todos, si no se indica la cláusula WHERE.

A continuación, presentamos algunos ejemplos que demuestran la funcionalidad básica de la instrucción DELETE usando la sintaxis mínima requerida.

a. Utilizar DELETE sin la cláusula WHERE

En el ejemplo siguiente se eliminan todas las filas de la tabla **SuppliersHistory** de la base de datos Northwind porque no se utiliza una cláusula WHERE para limitar el número de filas eliminadas.

```
DELETE FROM SuppliersHistory
```

b. Usar la cláusula WHERE para eliminar un conjunto de filas

En el ejemplo siguiente se eliminan todas las filas de la tabla **Shippers** de la base de datos Northwind en las que el valor de la columna **ShipperID** es superior a 5.

```
DELETE FROM Shippers  
WHERE ShipperID > 5
```

En el siguiente ejemplo se muestra una cláusula **WHERE** más compleja. La cláusula **WHERE** define dos condiciones que deben cumplirse para determinar las filas que se van a eliminar. El valor de la columna **Country** debe estar comprendido entre 'UK' y 'USA' y el valor de la columna **Fax** debe ser NULL.

```
DELETE SuppliersHistory  
WHERE Country IN ('UK', 'USA')  
AND Fax IS NULL
```

c. Usar combinaciones y subconsultas en los datos de una tabla para eliminar filas de otra tabla

En los siguientes ejemplos se muestran dos maneras de eliminar filas de una tabla en función de los datos de otra tabla. En ambos ejemplos, se eliminan las filas de la tabla **SuppliersHistory** de la base de datos Northwind basándose en los productos discontinuados almacenadas en la tabla **Products**.

La primera instrucción DELETE muestra la solución de subconsulta y la segunda instrucción DELETE muestra la extensión de FROM de Transact-SQL para unir las dos tablas.

```
DELETE FROM SuppliersHistory
WHERE SupplierID IN (SELECT SupplierID
                     FROM Products
                     WHERE Discontinued = 1)
```

```
DELETE SuppliersHistory
FROM SuppliersHistory
INNER JOIN Products ON SuppliersHistory.SupplierID = Products.SupplierID
WHERE Discontinued = 1
```