4.4. Modificación y eliminación de tablas

Ya se ha estudiado el primer comando perteneciente al lenguaje de definición de datos y que se usa para crear las tablas, `CREATE TABLE`. Para modificar y eliminar un objeto que ya existe se usa la palabra reservada ALTER y DROP respectivamente. Por tanto, para modificar una tabla se usará `ALTER TABLE` y para eliminarla se empleará `DROP TABLE`. Estos comandos, que pertenecen al DDL, no tienen vuelta atrás, es decir, no permiten la acción `ROLLBACK`, ya que tienen un COMMIT implícito.

Figura 4.20. *La modificación de la estructura de datos debe hacerse, en la medida de lo posible, antes de incluir registros en las tablas. La edición de estas estructuras son tareas bastante complejas, ya que los datos están relacionados, y se deben adaptar los dominios que se van a modificar.*

Cuando se modifican las estructuras de las bases de datos es importante analizar cómo están los datos almacenados en las tablas, y si es posible, modificarlas sin que afecten a los datos, en cuyo caso serán necesarios procesos de exportación e importación posterior.

4.4.1. Modificar las columnas y las restricciones

Para modificar una tabla ya creada, sea para añadir, modificar o eliminar una columna, añadir, modificar o eliminar una restricción, se usa el comando `ALTER TABLE`.

Ten en cuenta

No debe haber confusión entre el comando ALTER y el comando UPDATE. ALTER pertenece al sublenguaje de definición de datos, el sublenguaje DDL, y se usa para modificar la estructura de un objeto. Sin embargo, UPDATE pertenece al sublenguaje de manipulación de los datos, DML, y se emplea para modificar un registro que se encuentra en una tabla.

La sintaxis del comando `ALTER TABLE` es la que sigue:

```
ALTER TABLE [esquema].tabla {ADD|MODIFY|DROP {definición_
columna|CONSTRAINT definición_restriccion}}
```

Si se desea añadir una columna a una tabla que ya existe, se usaría ALTER TABLE tabla ADD definición_columna, por ejemplo,

```
ALTER TABLE Edicion ADD numTiradas NUMBER(5);
```

Si se desea añadir una restricción, se utilizaría `ALTER TABLE tabla ADD CONSTRAINT definición_restriccion`, por ejemplo,

```
ALTER TABLE Edicion ADD CONSTRAINT ck_ejem2 CHECK(numTiradas>=0
and numTiradas<=99999);
```

Si se desea eliminar una columna, se emplearía `ALTER TABLE tabla DROP [COLUMN] nombre_columna`, por ejemplo,

```
ALTER TABLE Edicion DROP numTiradas;
```

Si se desea eliminar una restricción, se usaría `ALTER TABLE tabla DROP nombre_restriccion`, por ejemplo,

```
ALTER TABLE Edicion DROP CONSTRAINT ck_ejem2;
```

Si se desea modificar una columna, se usa el nombre de la columna y la sintaxis `ALTER TABLE tabla MODIFY nombre_columna nueva_definición_columna`, por ejemplo,

```
ALTER TABLE Edicion MODIFY numTiradas NUMBER(6) NOT NULL DEFAULT 0;
```

Si se desea modificar una restricción, se emplea el nombre de la restricción con la sintaxis `ALTER TABLE tabla MODIFY CONSTRAINT nombre_restriccion nueva_definición_restriccion`, por ejemplo,

```
ALTER TABLE Edicion MODIFY CONSTRAINT ck _ ejem2 CHECK(numTiradas>
=0 and numTiradas<999999);
```

Ahora se ve la utilidad de nombrar las restricciones a través de la cláusula CONSTRAINT dentro de los comandos `CREATE TABLE`.

Para cambiar el nombre de una columna o de una restricción se usa la cláusula RENAME o RENAME COLUMN dentro del comando `ALTER TABLE`. Su sintaxis es la siguiente: `ALTER TABLE tabla RENAME [COLUMN] nombre_actual TO nombre_nuevo;` Por ejemplo, para cambiar el nombre de la columna *numTiradas* se escribiría `ALTER TABLE Edicion RENAME numTiradas TO numeroTiradas;`

4.4.2. Eliminar y cambiar el nombre de una tabla

Para eliminar una tabla se usa el comando DROP TABLE con la sintaxis siguiente:

```
DROP TABLE [esquema].tabla [CASCADE CONSTRAINTS|PURGE];
```

Cuando se intenta eliminar una tabla se tiene que tener en cuenta si contiene registros en su interior. Con este comando se eliminan todos sus registros y su estructura, es decir, se elimina por completo y no hay vuelta atrás. Así que es importante prestar mucha atención cuando se ejecuta este comando.

Además, se tiene que tener en cuenta si los registros que contiene están siendo referenciados en otras tablas. Si es así, la tabla no puede eliminarse, a menos que añadamos la cláusula CASCADE CONSTRAINTS, en cuyo caso se eliminará la tabla completa y las referencias que existan a ella desde otras tablas (claves ajenas). Por ejemplo, para eliminar la tabla Ejemplar con todos sus registros y todas las referencias a ella, se ejecutaría el comando

```
DROP TABLE Ejemplar CASCADE CONSTRAINTS;
```

Cuando se elimina una tabla el espacio que ocupa no se libera inmediatamente, ya que este es un proceso que puede tardar, dependiendo de la arquitectura del servidor. Para exigir que el espacio se libere lo antes posible se usa la opción PURGE. Así, con el comando

```
DROP TABLE LineaVenta CASCADE CONSTRAINTS PURGE;
```

la tabla y sus relaciones se eliminan completamente y el espacio que ocupa se libera para uso posterior.

Para cambiar el nombre de cualquier objeto se usa el comando RENAME con la sintaxis `RENAME nombre_actual TO nombre_nuevo;`. Por ejemplo, para cambiar el nombre de la tabla *Ejemplar* por *LibroEjemplar* se escribiría `RENAME Ejemplar TO LibroEjemplar;`.

Con el comando TRUNCATE se eliminan los registros de la tabla, pero no la estructura en sí (por lo que sigue en el diccionario de datos sin tocar). Además, se libera el espacio que los registros de la tabla ocupan en memoria secundaria. Hay que tener cuidado con su uso, ya que no se puede deshacer con un ROLLBACK. El comando quedaría como sigue:

```
TRUNCATE LibroEjemplar;
```

4.4.3. Activar y desactivar las restricciones

Las restricciones de una tabla pueden desactivarse para que no lleven el control de sus condiciones. Para llevar esta acción se usa DISABLE CONSTRAINT en el comando `ALTER TABLE`, indicando el nombre de la restricción. Por ejemplo,

```
ALTER TABLE DISABLE CONSTRAINT ck_ejem2 CASCADE;
```

Con CASCADE se desactivan aquellas restricciones que dependan de esta restricción. Para activar de nueva la restricción se usaría `ALTER TABLE ENABLE CONSTRAINT ck_ejem2 CASCADE;`.

4.5. Lenguaje de manipulación de datos

El lenguaje de manipulación de datos se encarga de la gestión de los registros que contienen las tablas de un esquema. En los apartados anteriores, se han estudiado los comandos relativos a la creación, modificación y eliminación de las estructuras de las tablas. Ahora es turno de tratar su contenido. Estos comandos, que para recordarlos se podrían llamar comandos INUPDEL, son los que permiten insertar un registro, `INSERT`, proceso relativo al alta de un registro, actualizarlo o modificarlo, `UPDATE`, y eliminarlo, `DELETE`.

4.5.1. Inserción de registros

Una vez que una tabla ya está creada con el conjunto de columnas definidas y sus restricciones declaradas, ya está lista para aceptar registros en su interior. Cada vez que se inserta un registro es importante tener claro cuál es su campo clave y definirlo para que sea único para toda la vida de la base de datos. Cuando se inserta un registro en una tabla se debe cumplir todos los requisitos declarados en ella, los campos obligatorios no admitirán nulos, los campos únicos así serán, los registros con claves ajenas deben existir en la tabla que hace referencia, se deben cumplir las restricciones de validación, etcétera.

Figura 4.24. *El comando INSERT pertenece al lenguaje DML y permite introducir registros en una tabla. La operación INSERT añade un nuevo registro o fila al final de una tabla. Una vez que un registro se ha creado su campo clave no debe ser modificado.*

La entrada de datos puede ser directamente por usuarios profesionales a través de aplicaciones, usuarios finales que explotan software, lectura sensorial de hardware especializado, importaciones de datos de servidores remotos o cualquier otro medio y tecnología.

El comando que se usa para insertar un registro es `INSERT` y la sintaxis para una tabla con n columnas es la que sigue:

```
INSERT INTO tabla[(columnas_que_introducir)] VALUES (valor1 [, valor2, …, valorn]);
```

La parte de columnas_que_introducir es opcional y permite indicar cuáles son las columnas a las que hacen referencia los valores indicados tras la palabra VALUES. Si esta opción no se usa, será necesario indicar un valor para todas las columnas.

Las columnas que no incluyen valor se rellenan con el valor DEFAULT declarado en la definición de la columna. Si no se ha indicado ningún valor por defecto, se incluirá el valor NULL, siempre que la columna admita nulos.

En la Figura 4.25 se puede observar los registros que se pretende insertar en la tabla Asignatura. Supóngase que se crearon en el orden CodAsig, CodCF, NumHoras y Nombre.

CodAsig	CodCF	NumHoras	Nombre
1	1	165	Bases de datos
2	1	120	Lenguaje de marcas
3	2	90	Seguridad informática
4	1	110	Despliegue aplicaciones webs
5	2	90	Fundamentos de hardware
6	1	180	Acceso a datos

Figura 4.25. *Esta tabla asignatura tiene las columnas en otro orden: CodAsig, CodCF, NumHoras, Nombre.*

El comando para crear esta tabla podría ser:

```
CREATE TABLE Asignatura(
        CodAsig number(3),
        CodCF number(2) NOT NULL,
        NumHoras NUMBER(3),
        Nombre VARCHAR2(50) NOT NULL,
        CONSTRAINT pk_asig PRIMARY KEY(CodAsig),
        CONSTRAINT ck_asig CHECK(numHoras BETWEEN 10 AND 900)
);
```

El comando para insertar el primer registro sería:

```
INSERT INTO Asignatura VALUES(1,1,165,'Base de datos');
```

La inserción también podría haberse hecho declarando el orden tras Asignatura, por ejemplo,

```
INSERT INTO Asignatura(CodAsig, CodCF, numHoras, nombre)
VALUES(1,1,165,'Base de datos');
```

En esta ocasión no es necesario indicar el orden de las columnas, ya que coinciden con el orden con el que se crearon. Ahora bien, si el orden no es el mismo, se podría haber escrito

```
INSERT INTO Asignatura(CodAsig, nombre, CodCF, numHoras) VALUES
(1,'Base de datos',165,1);
```

También se usa cuando no se conoce el valor de una columna y esta admite nulos.

```
INSERT INTO Asignatura(CodAsig, CodCF, nombre)  VALUES(1,1,'Base
de datos');
```

Esta última inserción añade en la columna que no se ha indicado valor, numHoras, el valor por defecto o el valor NULL si este no se ha declarado. El comando también podría haber sido el siguiente:

```
INSERT INTO Asignatura VALUES(1,1,NULL,'Base de datos');
```

Para la tabla de la Figura 4.25, estos serían los comandos que usar:

```
INSERT INTO Asignatura VALUES(1,1,165,'Base de datos');
INSERT INTO Asignatura VALUES(2,1,120,'Lenguaje de marcas');
INSERT INTO Asignatura VALUES(3,2,90,'Seguridad informática');
INSERT INTO Asignatura VALUES(4,1,110,'Despliegue aplicaciones webs');
INSERT INTO Asignatura VALUES(5,2,90,'Fundamentos de hardware');
INSERT INTO Asignatura VALUES(6,1,180,'Acceso a datos');
```

Generalmente, son las aplicaciones las que hacen las inserciones en una base de datos, conectándose a ella, usando una cuenta de usuario y con suficientes privilegios para poder operar en ella. Esos comandos se embeben con el código del programa.

Una vez que el usuario valida la información que introduce o importa, son las aplicaciones informáticas las que en su código embeben estas operaciones de inserción de datos. El SGBD controlará si la inserción es posible atendiendo a las restricciones inherentes y definidas en ellas.

4.5.2. Los campos autonuméricos

Las claves primarias no suelen estar sujetas a órdenes aleatorios de valores, al menos que sean datos como el DNI, número de la Seguridad Social, ISBN, código de barras, etc. Es muy habitual que los campos claves estén sujetos a una secuencia autonumérica. Si se observa la tabla de la Figura 4.25, los valores de CodAsig cumplen usa secuencia que comienza en 1 y se incrementa en cada inserción. Los SGBD disponen de mecanismos para automatizar esta cuenta y se usan en las inserciones de datos en las tablas. En versiones actuales de Oracle se puede usar la cláusula GENERATED ALWAYS AS IDENTITY, o en cualquier versión, crear una secuencia. Para ello, se usa el comando CREATE SEQUENCE con la siguiente sintaxis:

```
CREATE SEQUENCE nombreSecuencia INCREMENT BY valor | START WITH valor
| MAXVALUE valor | MINVALUE valor | CYCLE | NOCYCLE | CACHE | NOCACHE |
ORDER | NOORDER;
```

Las cláusulas más importantes son INCREMENT BY, para indicar el incremento de los valores, START WITH, el valor de inicio de la secuencia, MINVALUE, el valor mínimo cuando se comienza un nuevo ciclo, y MAXVALUE, para el valor máximo de la secuencia. La opción CYCLE permite que la secuencia comience de nuevo en caso de que llegue a su valor máximo. En caso contrario se usará NOCYCLE. En la cláusula de columna GENERATED se usan estos mismos parámetros.

El nombre de la secuencia debe ser único para cada tabla de la que quiere automatizar su columna autonumérica. Por tanto, se crearán tantas secuencias como columnas se quieran automatizar. Para la tabla Asignatura se podría crear la secuencia siguiente:

```
CREATE SEQUENCE seqAsignatura START WITH 1 INCREMENT BY 1
MAXVALUE 999 NOCYCLE;
```

Una vez que se ha creado una secuencia se puede usar en los comandos INSERT. Para ello, se utilizará el método NEXTVAL, que devuelve el valor actual de la secuencia y lo incrementa según el valor del incremento especificado con INCREMENT BY. Las inserciones anteriores hubieran quedado con el uso de la secuencia del siguiente modo:

```
INSERT INTO Asignatura VALUES(seqAsignatura.NEXTVAL,1,165,'Base de datos');
INSERT INTO Asignatura VALUES(seqAsignatura.NEXTVAL,1,120,'Lenguaje de marcas');
INSERT INTO Asignatura VALUES(seqAsignatura.NEXTVAL,2,90,'Seguridad informática');
INSERT INTO Asignatura VALUES(seqAsignatura.NEXTVAL,1,110,'Despliegue aplicaciones webs');
INSERT INTO Asignatura VALUES(seqAsignatura.NEXTVAL,2,90,'Fundamentos de hardware');
INSERT INTO Asignatura VALUES(seqAsignatura.NEXTVAL,1,180,'Acceso a datos');
```

Para comprobar el valor actual de la secuencia se puede ejecutar la siguiente consulta:

```
SELECT seqAsignatura.CURRVAL FROM dual;
```

Con respecto a otros sistemas gestores de bases de datos existen diferencias con respecto a las columnas autonuméricas. En SQL Server se usa IDENTITY y en MySQL, AUTO_INCREMENT en la definición de la columna dentro del comando `CREATE TABLE`. Por ejemplo,

```
CREATE TABLE Ciclo(CodCF int NOT NULL AUTO_INCREMENT PRIMARY KEY, nombre VARCHAR(255));
```

que en Oracle sería `CREATE TABLE Ciclo(CodCF NUMBER(3) GENERATED ALWAYS AS IDENTITY MINVALUE 1 MAXVALUE 999 INCREMENT BY 1 START WITH 1, Nombre VARCHAR2(255));`

4.5.3. Actualización de registros

Cuando se quieren actualizar los datos hay que analizar si esos cambios son buenos para la evolución de la base de datos, si deben estar permitidos y qué usuarios pueden modificarlos. Por ejemplo, cambiar el número de ejemplares que hay en una librería sobre sus productos puede ser un hecho fatal para la organización. Cuando los datos se transforman atendiendo a políticas establecidas, los valores antiguos pueden almacenarse en otros espacios con el fin de disponer de la evolución de estos. Este proceso se conoce como **historial de los datos.** Por ejemplo, interesaría en gran medida almacenar el historial de los precios de coste y de ventas de los productos de una empresa. Además, si, por ejemplo, un libro se vendió a 29 euros el año pasado y ahora cuesta 25, las ventas que se hicieron no pueden estar relacionadas con el nuevo precio, sino que deben seguir con su precio de venta de 29 euros.

Para modificar los valores almacenados en los registros de las tablas se usa el comando `UPDATE` con la siguiente sintaxis:

```
UPDATE tabla SET columna = valor_nuevo [,columna=valor_nuevo] [WHERE condicion];
```

Después de la cláusula SET se pueden poner tantas asignaciones como se quieran. La cláusula WHERE se usa para actualizar unas columnas si se cumple cierta condición. Siguiendo con el ejemplo anterior y si se quisiera cambiar el valor 165 por 243, es decir, que la asignatura Base de datos sea de 243 horas en vez de 165, se haría

```
UPDATE Asignatura SET numHoras=243 WHERE CodAsig=1;
```

También se podría haber filtrado por el nombre en vez de por el código.

```
UPDATE Asignatura SET numHoras=243 WHERE nombre='Base de datos';
```

Al usar el nombre se debe escribir todos los caracteres iguales a como se encuentra en la tabla. Por ejemplo, el siguiente comando no actualiza ningún registro, ya que no existe ningún nombre con 'Bases de datos'.

```
UPDATE Asignatura SET numHoras=243 WHERE nombre='Bases de datos';
```

Si se hubiera escrito el comando sin la condición, `UPDATE Asignatura SET numHoras=200;`, se habría actualizado toda la columna con el valor 243. También se puede actualizar una columna usando el valor de la propia columna. Por ejemplo, el siguiente comando incrementa en un 20 % el número de horas de aquellas asignaturas que tienen actualmente menos de 100 horas.

```
UPDATE Asignatura SET numHoras=numHoras*1.20 WHERE numHoras<100;
```

4.5.4. Borrado de registros

Eliminar los datos de una tabla siempre conlleva procesos de reflexión sobre su extracción y eliminación definitiva. En la mayoría de los casos, estos datos se exportan a tablas historiales. Además, casi seguro que los registros que se quieren eliminar están relacionados con otros y supone pérdida de información.

Figura 4.29. *Las operaciones de borrado definitivo deben estudiarse detenidamente. Se deben establecer políticas de seguridad para esos datos y mecanismos para el almacenamiento en repositorios como historiales, Data Warehouse o Big Data. El almacenamiento de datos que no forman la base de datos operacional está estrechamente relacionado con la base de la minería de datos y los Data Warehouse.*

El comando del que se dispone para eliminar registros es DELETE, y su sintaxis es como sigue:

```
DELETE [FROM] tabla [WHERE condicion];
```

Si no se establece una condición, se eliminarán todos los registros de la tabla. Por ejemplo, `DELETE Ejemplar;`. Para eliminar aquellos registros de Asignaturas con menos de 150 horas se escribiría `DELETE Asignaturas WHERE numHoras<150;`.