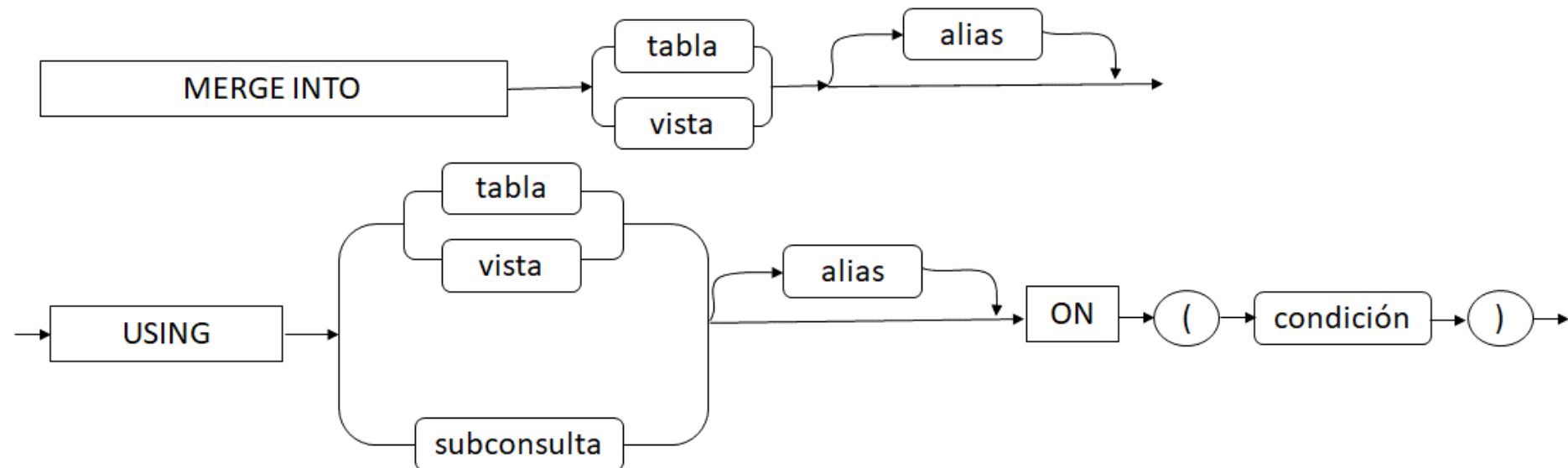


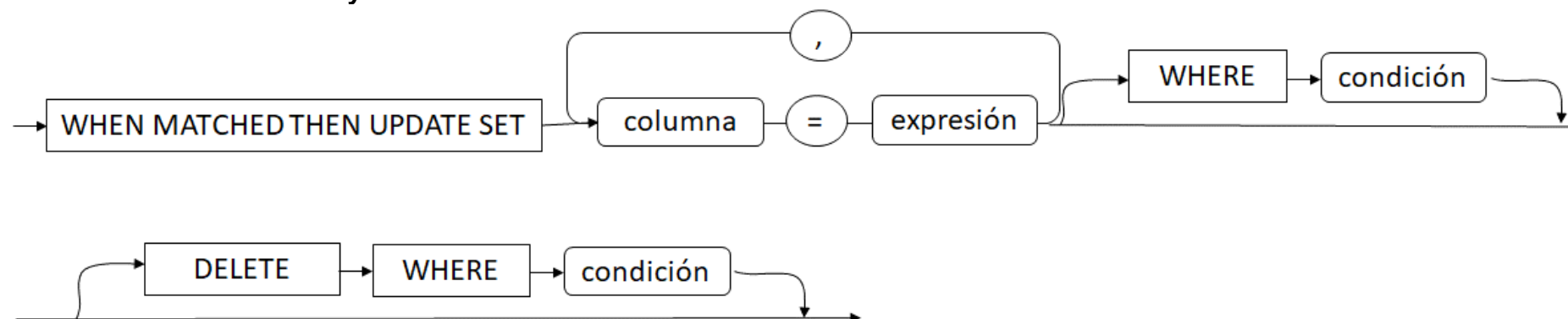
Sesión 2 – SENTENCIA MERGE, ALTER TABLE, IDENTITY, COMMIT, ROLLBACK y SAVEPOINT

MERGE para insertar y modificar en una tabla con una sola sentencia

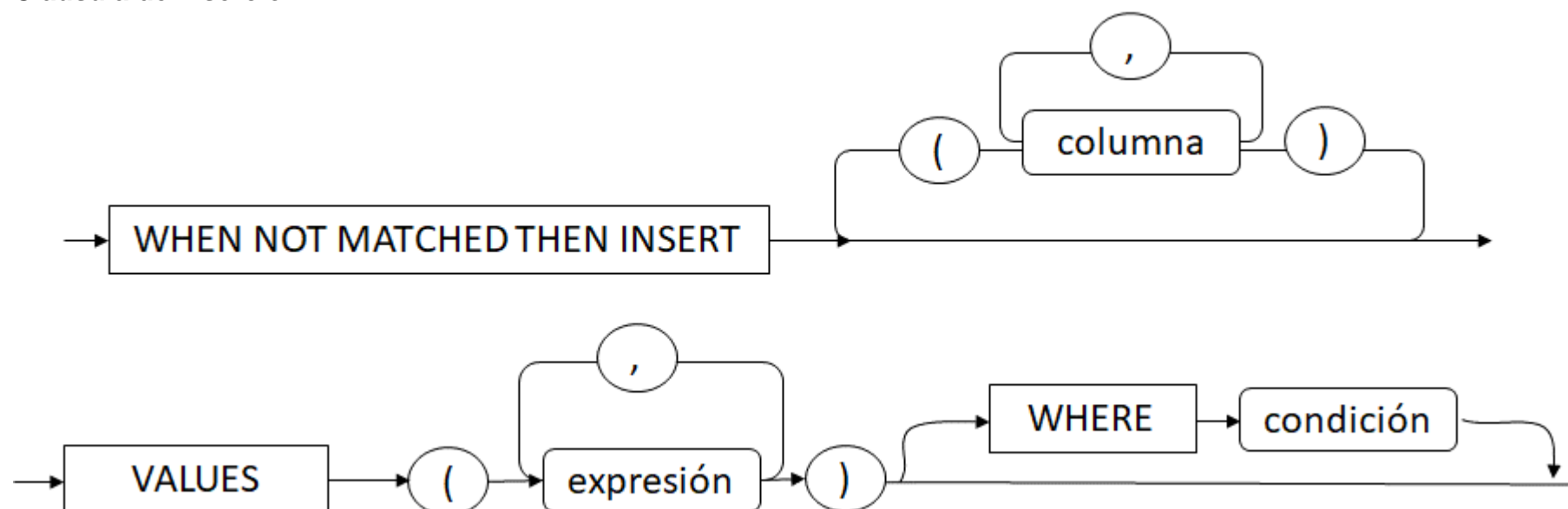
La sentencia MERGE se utiliza para seleccionar filas de una o más tablas origen, y modificarlas, borrarlas, o insertar filas en otra tabla. Se pueden especificar condiciones para determinar si se debe insertar o modificar en la tabla destino. La gran ventaja que tiene esta sentencia es que permite evitar múltiples acciones de inserción (INSERT), modificación (UPDATE), y borrado (DELETE) en determinados casos. Además, el gestor de la base de datos es más eficiente ejecutando una sola sentencia MERGE que sentencias INSERT y UPDATE separadas. Esto es debido a que la sentencia MERGE está preparada para manejar conjuntos de datos más que filas individuales.



Cláusula de modificación y borrado:



Cláusula de inserción:



Veamos el significado de las distintas partes de la sentencia:

- Cláusula INTO: La usaremos para definir la tabla sobre la que realizaremos las acciones de inserción, modificación, o borrado.
- Cláusula USING: Especificará el origen de los datos que aplicarán sobre la tabla destino.
- Cláusula ON: Indicará las condiciones que, una vez evaluadas como verdadero o falso, provocarán que la sentencia MERGE inserte o modifique en la tabla destino. Si la evaluación de la condición de esta cláusula es verdadera, MERGE modificará los datos que la cumplan. Si la condición se evalúa como falso, se insertarán los datos.
- Cláusula de modificación: Detallaremos las columnas que hay que cambiar en la tabla destino si se cumplen las condiciones de la cláusula ON, diciendo para cada columna qué valor sustituirá al actual.
- Cláusula de inserción: Indicaremos qué columnas de la tabla destino tendrán valores que queremos sean insertados en ella, y qué valores tomarán en las nuevas filas.

Ejemplo:

Tenemos estas dos tablas, que representan los precios en monedas virtuales que valen distintos artículos de un juego. La primera es la tabla actual, y la siguiente la tabla nueva.

Tabla_actual:

Articulo_ID	Articulo_precio
Espada	1.000
Hacha	2.500
Anillo de poder	5.000
Amuleto de vida	8.000

Tabla nueva:

Articulo_ID	Articulo_precio	Articulo_borrar
Espada	1.200	No
Anillo de poder	NULL	Sí
Amuleto de vida	8.000	No
Arco	10.000	No

Construyamos una sentencia MERGE que actualice la tabla de artículos actual, realizando de una sola vez estas tres acciones:

1. Modificar los precios de aquellos artículos que estén en las dos tablas.
2. Borrar las filas que en la tabla nueva indique un valor “Sí” en la columna Articulo_borrar.
3. Insertar los artículos nuevos que no estén en la tabla actual.

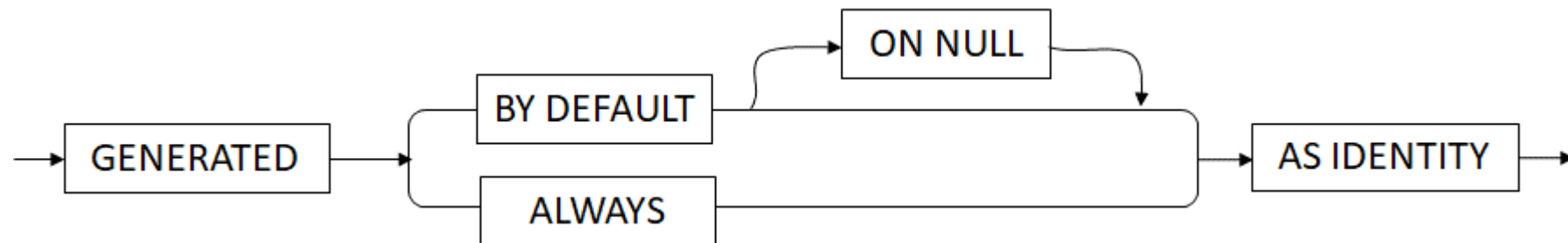
```

MERGE INTO Tabla_actual actual
USING Tabla_nueva nueva
ON (actual.Articulo_ID = nueva.Articulo_ID)
WHEN MATCHED THEN
  UPDATE SET
    Articulo_precio = nueva.Articulo_precio
  DELETE WHERE (nueva.Articulo_borrar = 'Sí')
WHEN NOT MATCHED THEN
  INSERT (Articulo_ID,Articulo_precio) VALUES (nueva.Articulo_ID, nueva.Articulo_precio);

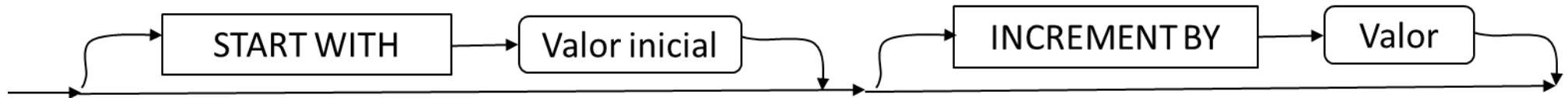
```

IDENTITY para incluir columnas con valores que se pueden incrementar automáticamente en las tablas

Las columnas IDENTITY, también llamadas autonuméricas, las utilizaremos para crear valores que el gestor de la base de datos incremente y gestione de manera automática. Al realizar inserciones de nuevos datos, será el gestor de la base de datos quien incremente el nuevo valor para la fila que estamos incluyendo. Realmente IDENTITY es una característica especial de los tipos de datos numéricos. La estructura sintáctica es ésta:



- **BY DEFAULT:** Nos permite utilizar Identity si la columna no se hace referencia en una sentencia insert (igual que ALWAYS), pero si se hace referencia a la columna, el valor especificado se utilizará en lugar de la identidad. El intento de especificar el valor NULL en este caso se traduciría en un error, ya que las columnas de identidad son siempre NOT NULL.
- **ALWAYS:** Indica que no será necesario introducir un valor para esta columna en una sentencia insert. Por el contrario, indicar un valor, aunque sea Null, producirá un error de ORACLE.
- **[ON NULL]:** Si añadimos esta cláusula opcional a BY DEFAULT, cuando insertemos un valor NULL para la columna no fallará, y añadirá un valor automático nuevo. En cambio, si se especifica un valor nuevo, será ése el que se incluirá en la tabla.



Al definir una columna como IDENTITY podemos utilizar las siguientes opciones:

- **START WITH initial_value:** Controla el valor inicial que usará la columna tipo identity. El valor inicial por defecto es 1.
- **INCREMENT BY interval_value:** Define el intervalo que el gestor dejará entre dos valores que genere. Por defecto será 1. Por ejemplo, si definimos un intervalo de 10 y un valor inicial de 10 también, la serie generada será 10, 20, 30, etc.

Las columnas autonuméricas las utilizaremos principalmente en estos casos:

- 1) Para crear claves primarias automáticas en tablas de datos en las que es difícil (o imposible) encontrar una combinación de columnas que nos las produzca.
- 2) Para crear claves primarias en tablas que tienen claves primarias complejas (compuestas por varias columnas) o de mucho tamaño (cadenas largas). Eso producirá que las claves ajenas que apunten a esas tablas tengan menor coste computacional en los joins, y menos coste de almacenamiento.

A estas columnas se les suele denominar claves subrogadas (surrogate keys en inglés).

Ejemplo:

Tenemos la siguiente sentencia que crea una tabla que representa los jugadores de un juego ON – LINE.

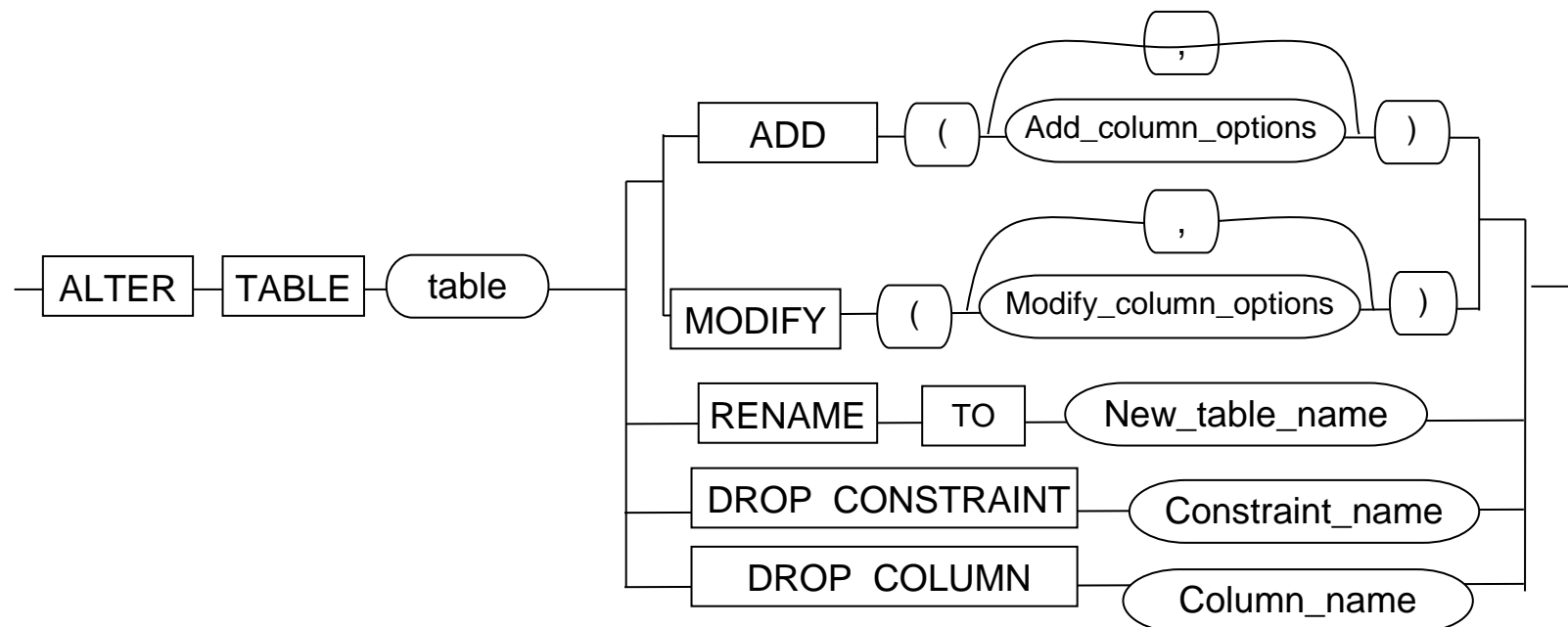
```
CREATE TABLE jugadores (  
  jugador_id      VARCHAR2(150) NOT NULL,  
  jugador_nombre  VARCHAR2(50) NOT NULL,  
  jugador_fecha_alta  DATE NOT NULL,  
  jugador_activo   INT DEFAULT 1 NOT NULL,  
  CONSTRAINT pk_jugadores PRIMARY KEY ( jugador_id ));
```

La columna JUGADOR_ID es la clave primaria, pero fijaos que es larga, y tiene 150 caracteres. Eso producirá que para cualquier tabla que tome claves ajenas desde ella, tengamos que incluir una columna de esa longitud. Modifiquémosla incluyendo una columna tipo IDENTITY:

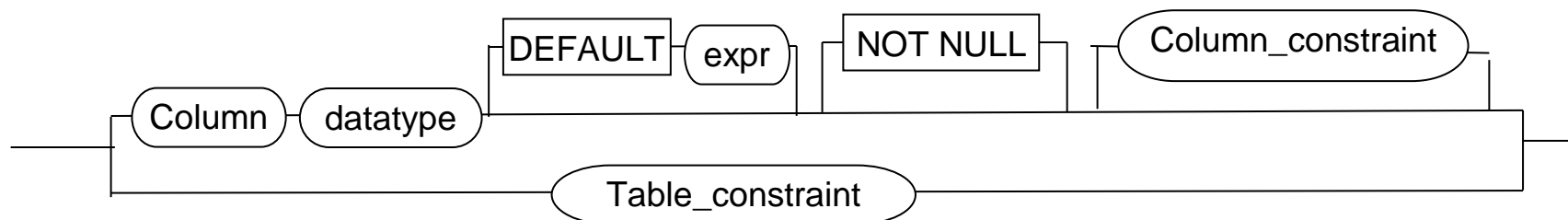
```
CREATE TABLE jugadores (  
  jugador_id      VARCHAR2(150) NOT NULL,  
  jugador_sk      INT GENERATED ALWAYS AS IDENTITY,  
  jugador_nombre  VARCHAR2(50) NOT NULL,  
  jugador_fecha_alta  DATE NOT NULL,  
  jugador_activo   INT DEFAULT 1 NOT NULL,  
  CONSTRAINT UNI_jugadores UNIQUE ( jugador_id ),  
  CONSTRAINT PK_jugadores PRIMARY KEY ( jugador_sk )  
);
```

ALTER TABLE para modificar la estructura y restricciones de una tabla

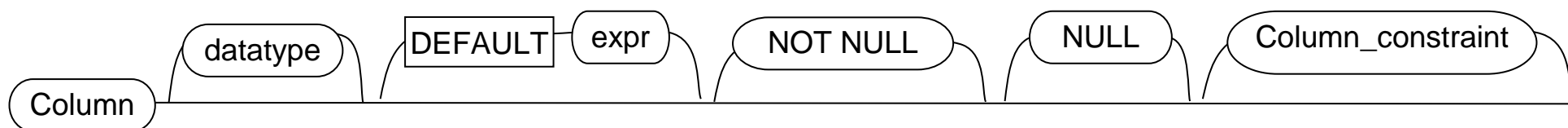
Después de crear las tablas se puede modificar su estructura con la sentencia ALTER TABLE. Nos permite añadir o borrar una nueva columna a una tabla, así como añadir o borrar restricciones a las columnas de la tabla.



Add_column_options



Modify_column_options



Ejemplos:

```
ALTER TABLE JUGADORES ADD Jugador_fnacimiento date;
```

```
ALTER TABLE JUGADORES MODIFY jugador_nombre null;
```

Suponiendo que existe una tabla NACIONALES con una columna llamada JUGADOR_ID. Para que sea FK hacia jugadores:

```
ALTER TABLE NACIONALES ADD constraint FK_jugadornacionales foreign key (JUGADOR_ID) references JUGADORES;
```

COMMIT, ROLLBACK y SAVEPOINT: concepto de transacción

TRANSACCIÓN:

- Conjunto de sentencias SQL que son tratadas por el SGDBR como unidad. **O todas son validadas o ninguna es validada.**
- El inicio lo marca la primera sentencia DML (Data Manipulation Language, es decir, inserts, updates, etc.) desde el último commit o rollback.
- *COMMIT*: hace permanentes los cambios producidos por la transacción.
- *ROLLBACK*: anula los cambios hechos por la transacción.
- Los cambios hechos por una transacción pendiente de COMMIT o ROLLBACK son visibles sólo por la sesión que ejecutó la transacción.
- Las sentencias DDL (Data Definition Language, es decir, CREATE, ALTER TABLE etc) llevan commit implícito.
- Si se produce una violación de constraint o cualquier otro error, el SGBDR hace un ROLLBACK automático.

COMMIT, ROLLBACK y SAVEPOINT: concepto de transacción

Update Clientes set domicilio = null
where codcli = '4536';

En este momento este usuario ve la dirección del cliente a nulo, pero el resto de sesiones todavía la ve sin cambios

Update facturas set Euros = pts*166.337
where pts is not null;

Delete from facturas
where Euros is null;

Si violasen alguna constraint o hubiese algún error la transacción Ejecutaría un rollback automático y terminaría

Commit;


La transacción se ha consolidado y ya todos ven la dirección a nulo. Ya no tiene posibilidad de rollback

COMMIT, ROLLBACK y SAVEPOINT: concepto de transacción

Update Clientes set domicilio = null
where codcli ='4536';

Update facturas set Euros = pts*166.337
where pts is not null;


Es el usuario el que provoca el rollback, con que se anulan todas las operaciones pendientes y comienza una nueva transacción



rollback;

Delete from facturas
where Euros is null;

La transacción se ha consolidado pero SÓLO se produce el borrado, los dos Updates anteriores ya estaban anulados



Commit;

COMMIT, ROLLBACK y SAVEPOINT: concepto de transacción

La sentencia SavePoint permite dividir una transacción grande en varios trozos. De esta forma si se produce un error no tenemos que deshacer toda la transacción, sino sólo hasta un save point

Update

insert;

Insert;

....

SavePoint P1

Definimos SavePoint



insert ...;

Delete

Rollback to P1

Deshacemos SÓLO las dos últimas sentencias

insert

COMMIT

NO incluye las sentencias entre SavePoint P1 y el ROLLBACK P1

Utilidad práctica de los conceptos vistos en la sesión

- Sentencias MERGE, columnas IDENTITY, y ALTER TABLE:

MERGE: Sirve para realizar simultáneamente sobre una tabla inserciones, borrados y modificaciones con una sola sentencia.

IDENTITY: Usaremos estas columnas para que el gestor de la base de datos autogestione su incremento.

ALTER TABLE: Nos permitirá modificar la estructura de las tablas.

- Commit, Rollback y Savepoint:

Nos dan un mecanismo para garantizar la integridad de la información en nuestra base de datos cuando efectuemos modificaciones de información delicadas o complejas.

Ejemplo:

Realización en un banco de una transferencia de una cuenta a otra.

- Sentencia 1: Obtener la cantidad a transferir.
- Sentencia 2: Modificar la cuenta origen, restándole el importe a transferir.
- Sentencia 3: Modificar la cuenta destino, sumándole el importe a transferir.

¿Qué ocurriría si la cuenta destino del dinero estuviese bloqueada, y no permitiese ingresos? La cuenta origen quedaría con el saldo mermado en la cantidad que queríamos transferir, y el dinero quedaría en el "limbo".

Usando la sentencia Rollback evitaríamos esa inconsistencia en los datos.