Administración de Sistemas Informáticos

DML: Manipulación de Datos

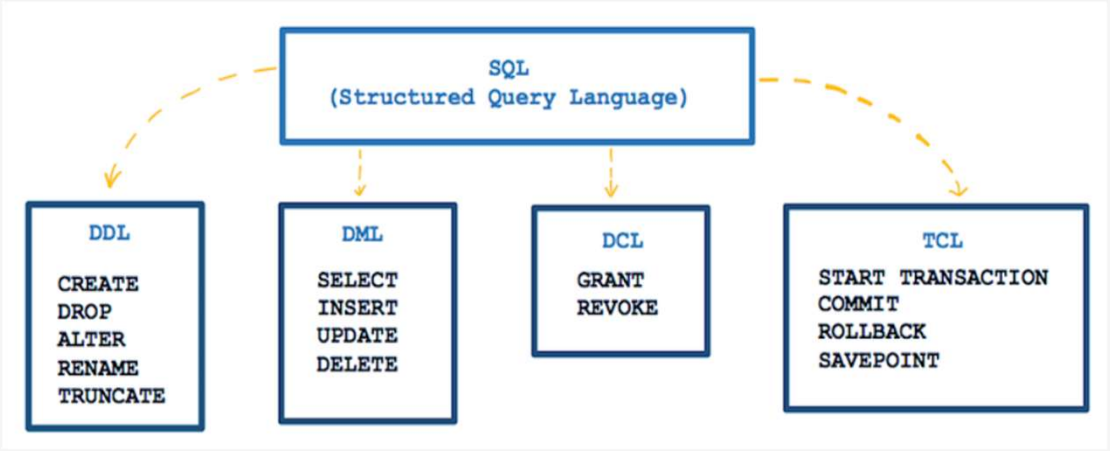
Modificación de los datos

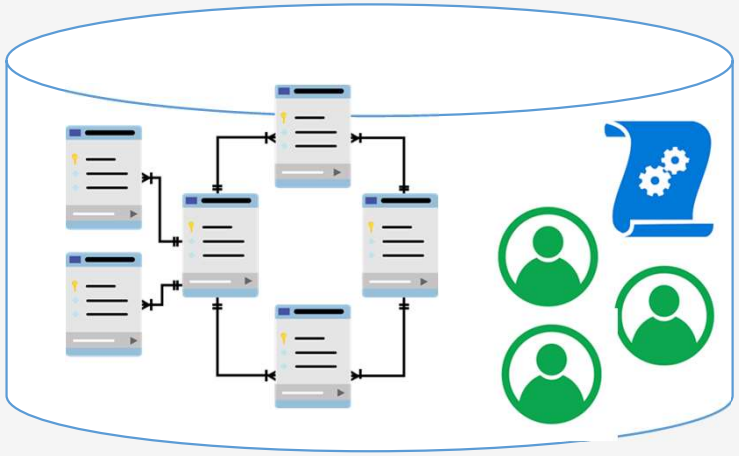
25/01/2023

El estándar SQL

***S**tructured **Q**uery **L**anguage* : Lenguaje estándar de definición y manipulación (y consulta) de bases de datos relacionales.

- *Características del Álgebra Relacional.*
- *Características del Cálculo Relacional de Tuplas.*
- *La versión que actualmente se encuentra más extendida es el SQL2 (ó SQL92).*





Modificación de Datos

Sentencias principales

INSERT: se utiliza para añadir registros a una tabla.

UPDATE: se utiliza para actualizar la información de uno o más registros de una tabla.

DELETE: se utiliza para eliminar uno o más registros de una tabla.

Inserción de datos

INSERT INTO tbl_name
[(col_name [, col_name] ...)]
{VALUES | VALUE} (value_list) [, (value_list)] ...

INSERT INTO tbl_name
SET *assignment_list*

INSERT INTO tbl_name
[(col_name [, col_name] ...)]
SELECT ...
value: {expr | DEFAULT}
value_list: value [, value] ...

assignment_list: col_name = value

```
INSERT INTO alumno
(nombre,DNI)
VALUES
        ('Kim',‘53200201Z’),
        (‘Axel’,‘13238299K’)
;
```

```
INSERT INTO alumno
SET
        nombre = 'Kim',
        DNI = ‘53200201Z ;
```

```
INSERT INTO clientes
(Nombre, Ciudad, Pais)
  SELECT SName, City,
Country
    FROM Suppliers
    WHERE Country='Germany';
```

Actualización y eliminación de datos

UPDATE table_reference
SET *assignment_list*
[**WHERE** where_condition]
[**ORDER BY** ...]
[LIMIT row_count]

value: {expr | DEFAULT}
assignment: col_name = value
assignment_list: assignment [, assignment] ...
value: {expr | DEFAULT}
value_list: value [, value] ...

```
UPDATE alumno
SET
        NOTA = 8.0
WHERE
        CURSO = '1SIT';
```

DELETE FROM table_reference
[**WHERE** where_condition]
[**ORDER BY** ...]
[LIMIT row_count]

```
DELETE FROM alumno
WHERE
        MATRICULA = false;
```

Actualizaciones y Eliminaciones con Integridad Referencial

CLAUSULAS CONSTRAINT SOBRE FOREING KEYS

• **ON DELETE y ON UPDATE**: Nos permiten indicar el efecto que provoca el borrado o la actualización de los datos que están referenciados por claves ajenas. Las opciones que podemos especificar son las siguientes:

- RESTRICT: Impide que se puedan actualizar o eliminar las filas que tienen valores referenciados por
- claves ajenas. Es la opción por defecto en MySQL.
- CASCADE: Permite actualizar o eliminar las filas que tienen valores referenciados por claves ajenas.
- SET NULL: Asigna el valor NULL a las filas que tienen valores referenciados por claves ajenas.
- NO ACTION: Es una palabra clave del estándar SQL. En MySQL es equivalente a RESTRICT.

Actualizaciones y Eliminaciones con Integridad Referencial

Una vista es una especie de “tabla virtual”, que muestra los resultados de la ejecución de una sentencia SELECT. Una vista también se define como una query almacenada (stored query)
Las vistas son de sólo lectura
Las vistas siempre están actualizadas porque se generan dinámicamente cuando se necesitan

Las vistas se usan para:

- ✓ Restringir el acceso de los usuarios al contenido completo de una/s tabla/s
- ✓ Ocultar la estructura interna de la base de datos
- ✓ Evitar la duplicación de información
- ✓ Tener una forma cómoda de acceder a joins que hagamos con frecuencia

RMS

UD5

Vistas

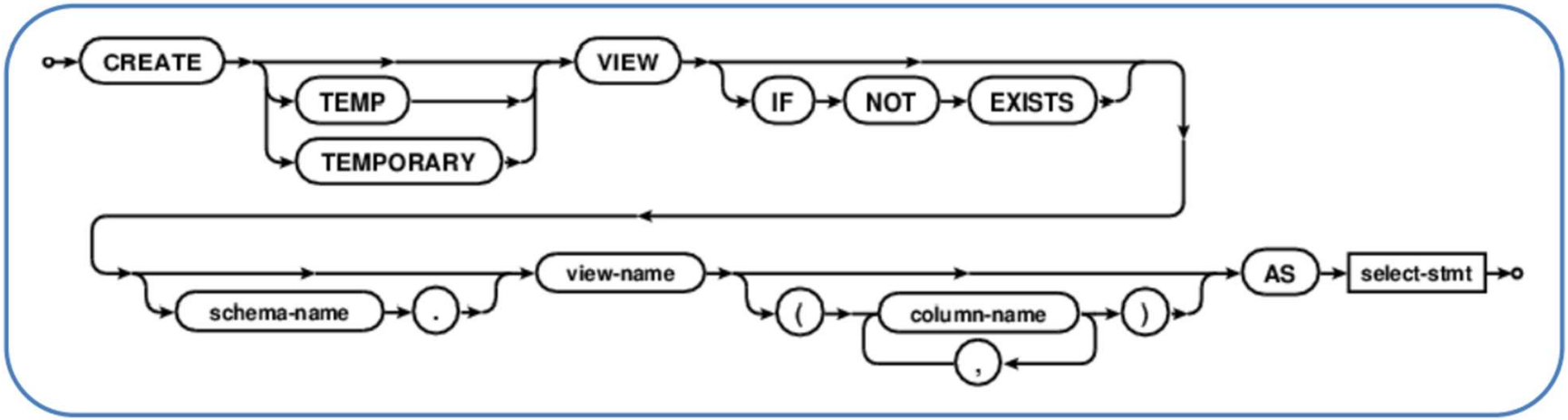
01/03/2023

Vistas

Una vista es una especie de "tabla virtual", que muestra los resultados de la ejecución de una sentencia SELECT. Una vista también se define como una query almacenada (stored query)

- Las vistas son de sólo lectura
- Las vistas siempre están actualizadas porque se generan dinámicamente cuando se necesitan
- Las vistas se usan para:
  - ✓ Restringir el acceso de los usuarios al contenido completo de una/s tabla/s
  - ✓ Ocultar la estructura interna de la base de datos
  - ✓ Evitar la duplicación de información
  - ✓ Tener una forma cómoda de acceder a joins que hagamos con frecuencia

Vistas : Definición y ejemplos




```
CREATE VIEW cantantes AS SELECT * FROM artists;
```

```
CREATE TEMP VIEW cantantes2 AS SELECT ArtistId AS ID, Name AS Nombre FROM artists;
```

```
CREATE VIEW empleados AS
        SELECT EmployeeId AS ID, FirstName || " " || LastName AS "Nombre Completo",
               Email "Correo electronico"
        FROM employees;
```

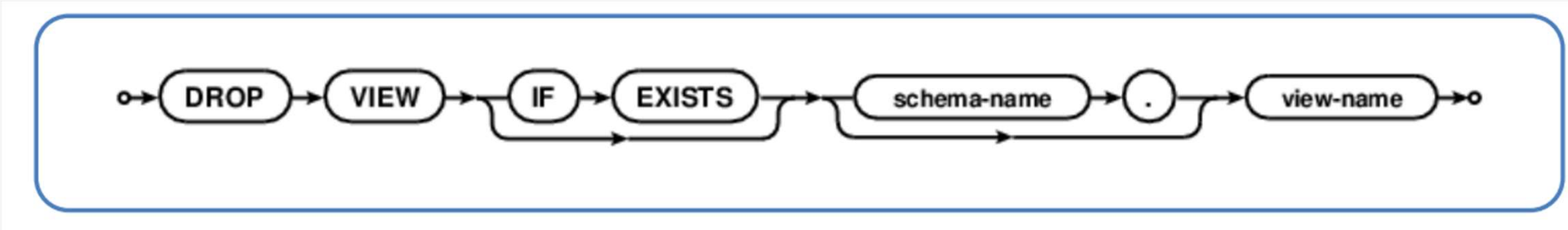
Vistas : Ejemplos (II)

```
CREATE VIEW canciones AS SELECT * FROM tracks
                         WHERE unitprice > 1 ORDER BY Name;
```

```
CREATE VIEW albumesLargos AS
        SELECT albums.Title AS "Titulo Album", COUNT(*) AS NumCanciones
        FROM tracks INNER JOIN albums ON Albums.AlbumId = tracks.AlbumId
        GROUP BY tracks.AlbumId HAVING NumCanciones >= 10
        ORDER BY numcanciones DESC;
```

```
CREATE VIEW canciones2 AS
        SELECT trackid, tracks.name, albums.Title AS album,
               media_types.Name AS media, genres.Name AS genres
        FROM tracks
             INNER JOIN albums ON Albums.AlbumId = tracks.AlbumId
             INNER JOIN media_types ON media_types.MediaTypeId = tracks.MediaTypeId
             INNER JOIN genres ON genres.GenreId = tracks.GenreId;
```

Vistas : Borrado




```
DROP VIEW empleados;
```

```
DROP VIEW IF EXISTS empresa1.empleados;
```

DROP VIEW no ocasiona la eliminación de los registros resultantes de la misma.

RMS

UD5

Transacciones

01/03/2023

Concepto

Una ***transacción*** es un conjunto de instrucciones SQL que se ejecutan como una unidad, de manera atómica.

- Cuando una transacción tiene éxito, todas las operaciones que la componen se guardan en la base de datos de manera definitiva.
- Si no, no se guarda nada

- **Propiedades ACID**:

  a) **A**tomicity (Atomicidad). Asegura que se realizan todas las operaciones o ninguna
  b) **C**onsistency (Consistencia). Sólo se permiten escribir datos correctos y coherentes con los existentes. Relacionada con la integridad referencial
  c) **I**solation (Aislamiento). Asegura que la operación no afecta a otras. Bloqueos de información para preservar la coherencia
  d) **D**urability (Durabilidad). Una vez finalizada con éxito, los cambios se hacen permanentes

Transacciones en MySQL

MySQL proporciona transacciones a través del motor de almacenamiento InnoDB. En general, una transacción puede terminar con:

- Éxito (COMMIT). Se ejecutan todas las operaciones y se hacen permanentes
- Fallo (ROLLBACK). No se ejecuta ninguna operación y todo queda como antes de comenzar las operaciones

Pasos generales

1. Iniciar la transacción
2. Realizar las operaciones de actualización, inserción y/o borrado correspondientes
3. Si se quieren conservar los cambios, finalizar la transacción (*COMMIT*)
4. Si sucede algún problema con la ejecución de las operaciones, cancelar los cambios (*ROLLBACK*)

Transacciones en MySQL

Sentencias disponibles

MySQL proporciona las siguientes sentencias para usar transacciones:

- ✓ **START TRANSACTION** o BEGIN para comenzar una transacción
- ✓ **COMMIT** para finalizar la transacción actual y hacer los cambios persistentes
- ✓ **ROLLBACK** para deshacer los cambios realizados en la transacción actual
- ✓ **SET autocommit** para activar/desactivar las transacciones automáticas en la sesión actual

  autocommit es una variable de sesión, que puede ser modificada temporalmente para la sesión actual o en el fichero de configuración (my.cnf – my.ini)

```
SET autocommit = {0 | 1};
```

Por defecto, autocommit está activado. Si se desactiva (autocommit = 0), se debe hacer un COMMIT explícito para hacer permanentes los cambios

Transacciones en MySQL

- Para deshabilitar el modo autocommit = 1 para una transacción (conjunto de operaciones) concreta, sin tocar el valor de la variable, se usa la sentencia START TRANSACTION

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

- Comenzar una transacción provoca que:
  - ✓ Cualquier transacción pendiente sea confirmada/ejecutada
  - ✓ Los bloqueos establecidos con LOCK TABLES se eliminen, como si se hubiese ejecutado UNLOCK TABLES
- Las transacciones no confirmadas (*commited*), no son registradas en los ficheros binarios de log

Transacciones

Sentencias que no se pueden deshacer (rolled back)

- Estas sentencias incluyen las *sentencias DDL* (Data Definition Language), es decir, sentencias de creación y borrado (**`CREATE, DROP`**) de bases de datos y tablas, así como de modificación (**`ALTER`**) de tablas y procedimientos almacenados
- Las transacciones deben diseñarse con cuidado, para intentar no incluir este tipo de sentencias ➢

- Recomendación: sólo sentencias de manipulación de datos (DML – SELECT, INSERT, UPDATE y DELETE

Transacciones

Sentencias que provocan un commit implícito (automático)

Confirman cualquier transacción pendiente en la sesión actual.La razón es que son operaciones que no se pueden deshacer

- ✓ Sentencias **DDL**: `CREATE, ALTER, DROP, RENAME, TRUNCATE`…
- ✓ Sentencias que implícitamente **usan o modifican tablas en la base de datos mysql**: `[ALTER | CREATE | RENAME | DROP] USER, GRANT, REVOKE, SET PASSWORD`
- ✓ Sentencias de **bloqueo de tablas**: `[LOCK | UNLOCK] TABLES`
- ✓ Sentencias de **control de transacciones:** START TRANSACTION, SET autocommit = 1, BEGIN
- ✓ Sentencias de **carga de datos:** `LOAD DATA INFILE`
- ✓ Muchas de ellas también provocan un commit implícito después de su ejecución.

Las transacciones NO se pueden anidar

Control de Transacciones

El motor InnoDB soporta las sentencias SQL

```
SAVEPOINT identifier
ROLLBACK [WORK] TO [SAVEPOINT] identifier
RELEASE SAVEPOINT identifier
```

SAVEPOINT: establece una etiqueta mediante un identificador

ROLLBACK TO SAVEPOINT: deshace los cambios hasta la etiqueta indicada, dentro de la transacción actual. Si no existe la etiqueta, devuelve un error:

```
ERROR 1305 (42000): SAVEPOINT identifier does not exist
```

RELEASE SAVEPOINT: elimina la etiqueta indicada de la transacción actual

Después de un `commit` o `rollback`, todos los savepoints son eliminados

Ejemplo de uso de SAVE POINTS

Crear una base de datos con la siguiente información

```sql
DROP DATABASE IF EXISTS test;
CREATE DATABASE test CHARACTER SET utf8mb4;
USE test;

CREATE TABLE producto (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  precio DOUBLE
);

INSERT INTO producto (id, nombre) VALUES (1, 'Primero');
INSERT INTO producto (id, nombre) VALUES (2, 'Segundo');
INSERT INTO producto (id, nombre) VALUES (3, 'Tercero');
```

Ejecutar las operaciones con :

autocommit = 1

Ejemplo de uso de SAVE POINTS

Crear una base de datos con la siguiente información

```sql
-- 1. Comprobamos las filas que existen en la tabla
SELECT *
FROM producto;

-- 2. Ejecutamos una transacción que incluye un SAVEPOINT
START TRANSACTION;
INSERT INTO producto (id, nombre) VALUES (4, 'Cuarto');
SAVEPOINT sp1;
INSERT INTO producto (id, nombre) VALUES (5, 'Cinco');
INSERT INTO producto (id, nombre) VALUES (6, 'Seis');
ROLLBACK TO sp1;
```

```sql
-- 3. ¿Qué devolverá esta consulta?
SELECT *
FROM producto;
```

RMS

UD5

Acceso concurrente a datos

06/03/2023

Exposición de la problemática

Cuando dos transacciones distintas intentan acceder concurrentemente a los mismos datos pueden ocurrir los siguientes problemas:

Dirty Read (Lectura sucia). Sucede cuando una segunda transacción lee datos que están siendo modificados por una transacción antes de que haga COMMIT.

Non-Repeatable Read (Lectura No Repetible). Se produce cuando una transacción consulta el mismo dato dos veces durante la ejecución de la transacción y la segunda vez encuentra que el valor del dato ha sido modificado por otra transacción.

Phantom Read (Lectura fantasma). Este error ocurre cuando una transacción ejecuta dos veces una consulta que devuelve un conjunto de filas y en la segunda ejecución de la consulta aparecen nuevas filas en el conjunto que no existían cuando se inició lo transacción.

Ejemplos

Dirty Read

Transacción 1	Transacción 2
`UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1;`	
	`SELECT saldo FROM cuentas WHERE id = 1;`
`ROLLBACK`	

Non-Repeatable Read

Transacción 1	Transacción 2
`SELECT saldo FROM cuentas WHERE id = 1;`	
	`UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1;`
`SELECT saldo FROM cuentas WHERE id = 1;`	

Ejemplos

Phantom Read

Transacción 1	Transacción 2
`SELECT SUM(saldo) FROM cuentas;`	
	`INSERT INTO cuentas VALUES (4, 3000);`
`SELECT SUM(saldo) FROM cuentas;`	

Niveles de Aislamiento

Controlan el nivel de bloqueo durante el acceso a los datos.

El estándar ANSI/ISO de SQL (SQL92) define cuatro niveles de aislamiento:

- ✓ **Read Uncommitted.** En este nivel no se realiza ningún bloqueo, por lo tanto, permite que sucedan los tres problemas
- ✓ **Read Committed.** En este caso los datos leídos por una transacción pueden ser modificados por otras transacciones, por lo tanto, se pueden dar los problemas Non-Repeteable Read y Phantom Read.
- ✓ **Repeatable Read**. En este nivel ningún registro leído con un SELECT puede ser modificado en otra transacción, por lo tanto, sólo puede suceder el problema del Phantom Read.
- ✓ **Serializable**. En este caso las transacciones se ejecutan unas detrás de otras, sin que exista la posibilidad de concurencia.

El nivel de aislamiento que utiliza InnoDB por defecto es Repeatable Read.

Niveles de Aislamiento

Tabla resumen

Nivel	Dirty Read (Lectura sucia)	Non-Repeatable Read (Lectura No Repetible)	Phantom Read (Lectura fantasma)
Read Uncommitted	Es posible	Es posible	Es posible
Read Committed	-	Es posible	Es posible
Repeatable Read	-	-	Es posible
Serializable	-	-	-

Nivel de aislamiento en curso: existen variables global y de sesión : `@@transaction_isolation`.

-- Variable global
```
SELECT @@GLOBAL.transaction_isolation;
```

-- Variable de sesión
```
SELECT @@SESSION.transaction_isolation;
```

Ejemplo de uso de niveles de aislamiento

Crear base de datos , desde **Terminal A**

```sql
DROP DATABASE IF EXISTS test;
CREATE DATABASE test CHARACTER SET utf8mb4;
USE test;

CREATE TABLE cuentas (
    id INTEGER UNSIGNED PRIMARY KEY,
    saldo DECIMAL(11,2) CHECK (saldo >= 0)
);

INSERT INTO cuentas VALUES (1, 1000);
INSERT INTO cuentas VALUES (2, 2000);
INSERT INTO cuentas VALUES (3, 0);
```

-- 1. Configuramos que en esta sesión vamos a utilizar el nivel de aislamiento READ UNCOMMITTED

```sql
SET SESSION TRANSACTION ISOLATION LEVEL
READ UNCOMMITTED;
```

-- 2. Ejecutamos una transacción para transfereir dinero entre dos cuentas

```sql
START TRANSACTION;
UPDATE cuentas SET saldo = saldo - 100
WHERE id = 1;
```

La transacción que estamos ejecutando en el terminal A todavía no ha finalizado, porque no hemos ejecutado COMMIT ni ROLLBACK.

Ejemplo de uso de niveles de aislamiento

Desde **Terminal B** ejecutar :

```
-- 1. Seleccionamos la base de datos
USE test;
-- 2. Configuramos que en esta sesión vamos a utilizar el nivel de aislamiento READ UNCOMMITTED
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
-- 3. Iniciamos una transacción y observamos los datos que existen en la tabla cuentas
START TRANSACTION;
SELECT * FROM cuentas WHERE id = 1;
```

Desde **Terminal A** ejecutar :

```
-- 3. Deshacemos las operaciones realizadas en la transacción
ROLLBACK;
```

Desde **Terminal B** ejecutar :

```
-- 4. Observamos los datos que existen en la tabla cuentas
SELECT * FROM cuentas WHERE id = 1;
```

RMS

UD5

Ejercicios

06/03/2023

Ejercicios

Edición de datos

Realice las siguientes operaciones sobre la base de datos jardineria.

1. Inserta una nueva oficina en Almería.
2. Inserta un empleado para la oficina de Almería que sea representante de ventas.
3. Inserta un cliente que tenga como representante de ventas el empleado que hemos creado en el paso anterior.
4. Inserte un pedido para el cliente que acabamos de crear, que contenga al menos dos productos diferentes.
5. Actualiza el código del cliente que hemos creado en el paso anterior y averigua si hubo cambios en las tablas relacionadas.
6. Borra el cliente y averigua si hubo cambios en las tablas relacionadas.
7. Elimina los clientes que no hayan realizado ningún pedido.
8. Incrementa en un 20% el precio de los productos que no tengan pedidos.
9. Borra los pagos del cliente con menor límite de crédito.
10. Establece a 0 el límite de crédito del cliente que menos unidades pedidas tenga del producto 11679.
11. Modifica la tabla detalle_pedido para insertar un campo numérico llamado iva. Mediante una transacción, establece el valor de ese campo a 18 para aquellos registros cuyo pedido tenga fecha a partir de Enero de 2009. A continuación actualiza el resto de pedidos estableciendo el iva al 21.

Ejercicios

Edición de datos

12. Modifica la tabla detalle_pedido para incorporar un campo numérico llamado `total_linea` y actualiza todos sus registros para calcular su valor con la fórmula:

```
total_linea = precio_unidad*cantidad * (1 + (iva/100));
```

12. Borra el cliente que menor límite de crédito tenga. ¿Es posible borrarlo solo con una consulta? ¿Por qué?
13. Inserta una oficina con sede en Granada y tres empleados que sean representantes de ventas.
14. Inserta tres clientes que tengan como representantes de ventas los empleados que hemos creado en el paso anterior.
15. Realiza una transacción que inserte un pedido para cada uno de los clientes. Cada pedido debe incluir dos productos.
16. Borra uno de los clientes y comprueba si hubo cambios en las tablas relacionadas. Si no hubo cambios, modifica las tablas necesarias estableciendo la clave foránea con la cláusula ON DELETE CASCADE.

RMS

UD5

Bloqueos

06/03/2023