Manipulación y control de datos

1º DAW. Bases de datos. Colegio calasanz

Contenidos

- Lenguaje de manipulación de datos
- Inserción de datos:
  - INSERT
  - INSERT extendida
  - INSERT – SELECT
  - CREATE TABLE AS SELECT
- Modificación de datos:
  - UPDATE
- Eliminación de datos
  - DELETE
- Modificación y borrado de registros con relaciones
- Lenguaje de control de datos
  - Copias de seguridad
  - Control de acceso a los datos
  - Transacciones
- Acceso concurrente a los datos

Lenguaje de manipulación de datos

- Lenguaje de manipulación de datos → DML
- Permite, dentro de una BBDD:
  - Insertar tuplas → INSERT
  - Modificar tuplas → UPDATE
  - Eliminar tuplas → DELETE

Inserción de datos (1). INSERT (1)

- Sintaxis:

  INSERT [INTO] nombre_tabla [(nonmbre_columna, . . .)]

  VALUES ({expr | DEFAULT},...)

- Después del nombre de la tabla, de forma optativa, se pueden indicar las columnas donde se va a insertar la información
  - Si se especifican las columnas, la lista de valores (VALUES) a insertar, se asociarán correlativamente con los valores a las columnas indicadas.
    - Ej: INSERT INTO mascotas (Codigo, Nombre) VALUES (1, 'Paquitas'); → se asigna NULL a las columnas no especificadas
  - Si no se especifican las columnas, la lista de valores se escribirá conforme al orden de las columnas en la definición de la tabla → se deben introducir todas las columnas
    - Ej: INSERT INTO mascotas VALUES (2, 'Calcetines', 'Gato Común Europeo', '59932387L’);

Inserción de datos (II). INSERT (II)

- #INSERT con columnas con valores por defecto
  - Ej: se ha creado una tabla con la siguiente definición:
    - CREATE TABLE coches (matricula varchar(11) NOT NULL,
      Marca varchar(50) DEFAULT 'seat',
      Modelo varchar(100) DEFAULT NULL,
      PRIMARY KEY (matricula)) ENGINE=InnoDB DEFAULT CHARSET=latin1;
  - Ej: INSERT INTO coches VALUES ('1215 BCD','Toledo IDI', DEFAULT);
  - El valor DEFAULT asignará el valor por defecto a la tercera columna de la tabla vehículos → valor NULL.
- #INSERT Errónea
  - Sentencia INSERT con más campos en la lista de valores que el número de columnas especificadas (o número de columnas de la tabla)
    - El SGBD informará del error
  - Ej: INSERT INTO coches (Matricula,Modelo,Marca) VALUES ('4123 BFH','Ibiza');
    - ERROR 1136 (21S01): Column count doesn't match valué count at row 1

Inserción de datos (III). INSERT extendida

- Sentencia INSERT extendida
  - Permite, en algunos SGBD, introducir varias tuplas con la misma sentencia.
  - Sintaxis:

    INSERT [INTO] nombre_tabla [(nombre.columna,. . .)]

    VALUES ({expr | DEFAULT},...),(…),…
  - Ej: INSERT INTO coches (Matricula,Modelo,Marca)

    VALUES ('4123 BFH','Ibiza','Seat'),

    ('1314 FHD','Toledo','Seat'),

    ('3923 GJS','León','Seat');

Inserción de datos (IV). INSERT-SELECT (1)

- Es posible obtener un conjunto de datos mediante una SELECT y después insertarlo mediante INSERT.
- Sintaxis:

  INSERT [INTO] nombre_tabla [(nombre_columna,.,)]

  SELECT ... FROM ...
- Ej: insertar en la tabla Backup todos los vehículos (la tabla se ha creado previamente)
- INSERT INTO BackupVehiculos SELECT * FROM vehiculos;

Inserción de datos (V). INSERT-SELECT (II)

- SELECT debe devolver tantas columnas como columnas tenga la tabla donde se introduce la información.

- En el ejemplo anterior, la tabla BackupVehiculos tiene una estructura idéntica a la tabla vehículos.

- La sentencia SELECT puede ser tan compleja como se desee, usando filtros, agrupaciones, ordenaciones, etc.

Inserción de datos (V). CREATE TABLE AS SELECT

- Nos permite crear una tabla mediante una selección de datos mediante SELECT.
- Ej 1: CREATE TABLE coches_backup AS

  SELECT * FROM coches;
  - Crearía un backup con todos los datos de la tabla coches.
- Ej 2: CREATE TABLE coches_backup AS

  SELECT * FROM coches LIMIT 0;
  - Crearía una tabla vacía con estructura similar a la tabla coches → LIMIT 0 evita que se devuelvan tuplas.

Modificación de datos (1).

- UPDATE permite modificar el contenido de cualquier columna y de cualquier fila de una tabla.
- Sintaxis:

  UPDATE nombre_tabla SET nombre_col1=expr1 [, nombre_col2=expr2] ... [WHERE filtro]

- La actualización se realiza dando a las columnas nombre_col1, nombre_col2... Los valores expr1, expr2...
- Se actualizan todas las filas seleccionadas por el filtro indicado mediante la cláusula WHERE
  - Es idéntica a la que se ha utilizado para el filtrado de registros en la sentencia SELECT.

Modificación de datos (II).

- Ej: se desea actualizar el equipo de ‘Pau Gasol’ a 'Knicks'
  - UPDATE jugadores SET Nombre_equipo='Knicks’

    WHERE Nombre='Pau Gasol';
  - Query OK, 1 row affected (0.02 se)
  - Rows matched: 1 Changed: 1 Warmings: O
- El SGBD indica que el filtro seleccionó una fila y cambió 1 fila → en este caso, la columna Nombre_equipo de esa fila seleccionada.

Modificación de datos (III).

- Cambiar varias columnas al mismo tiempo.
  - Ej: UPDATE jugadores SET Nombre_equipo='Knicks', Peso=210 WHERE Nombre='Pau Gasol';
- Si se omite el filtro, el resultado es la modificación de todos los registros de la tabla
- Ejemplo que cambia el peso de todos los jugadores de la NBA de libras a kilos: UPDATE jugadores SET Peso=Peso*0.4535;

Eliminación de datos. DELETE

- DELETE permite eliminar tuplas.
- Sintaxis:
  DELETE FROM nombre_tabla [WHERE filtro]
- Se eliminan todas las filas seleccionadas por el filtro indicado mediante la cláusula WHERE
  - Es idéntica a la que se ha utilizado para el filtrado de registros en la sentencia SELECT.
- Ej: se desea borrar al jugador 'Jorge Garbajosa' de la BBDD:
  - DELETE FROM jugadores WHERE Nombre='Jorge Garbajosa';
    - Query OK, 1 row affected (0.01 sec)
  - Si se omite el filtro, el resultado es el borrado de todos los registros de la tabla:
    DELETE FROM jugadores;
    - Query OK, 432 rows affected (2.42 sec)

Modificación y borrado de registros con relaciones (I)

- No siempre se pueden borrar o modificar datos. Ejemplos:
  - Un cliente llama a una empresa para darse de baja como cliente, pero el cliente tiene algunos pagos pendientes. Si el operador de la BBDD intenta eliminar el registro (DELETE), el SGBD debería informar de que no es posible eliminar ese registro puesto que hay registros relacionados.
  - Se desea cambiar (UPDATE) el nombre de un equipo de la NBA (que es su clave primaria), ¿qué sucede con los jugadores? También habrá que cambiar el nombre del equipo de los jugadores, puesto que el campo NombreEquipo es una clave foránea.
- Cláusulas REFERENCES de la sentencia CREATE TABLE → crean las relaciones de clave foránea-clave primaria de alguna columna de una tabla.

Modificación y borrado de registros con relaciones (II)

- Referencias:

  REFERENCES nombre_tabla [(nombre_columna,. ..)]

  [ON DELETE opción_referencia]

  [ON UPDATE opción_referencia]

- opción_referencia:
  - CASCADE I SET NULL I NO ACTION

- Se recomienda repasar la teoría vista en la UD “Creación, supresión y modificación de tablas”

Lenguaje de control de datos

- Lenguaje de manipulación de datos → DCL
- Permite, dentro de una BBDD:
  - Controlar el acceso a los datos
  - Controlar las transacciones
  - Manejar copias de seguridad
  - Etc.

Copias de seguridad

- Permiten almacenar todos los datos en “script”
  - Existen varias opciones para exportar los datos
- Se pueden recuperar/importar los datos en una BBDD utilizando el script anterior
- Los SGBD ofrecen diferentes mecanismos para realizar estas tareas.
  - Tareas programadas/manuales, gráficas/interfaz de comandos, etc.

Control de acceso a los datos (1)

- Permite definir los permisos de seguridad de los usuarios de la BBDD
- Los permisos se otorgan sobre los diferentes objetos de la BBDD → cada objeto puede tener uno o más permisos
- Conceder permisos:
  - Sintaxis:

    GRANT [CONNECT|RESOURCE|DBA|ALL PRIVILEGES|SELECT|UPDATE|INSERT|DELETE]

    ON <objeto>

    TO <lista_usuarios> [WITH GRANT OPTION];

Control de acceso a los datos (II)

- WITH GRANT OPTION permite a los usuarios cambiar permisos de acceso
  - Hay que tener cuidado a la hora de utilizar esta opción para no crear agujeros de seguridad.
- Eliminar permisos:
  - Sintaxis:

    REVOKE [CONNECT|RESOURCE|DBA|ALL PRIVILEGES|SELECT|UPDATE|INSERT|DELETE]

    FROM <lista_usuarios>

    [ON <lista_objetos>];

Transacciones (I)

- Una transacción es un conjunto de sentencias SQL que se tratan como una sola instrucción (atómica)
- Un SGBD actualiza múltiples datos a través de una transacción.
- Una transacción puede ser:
  - Confirmada (commit) → si todas las operaciones individuales se ejecutaron correctamente
  - Abortada (rollback) → si a la mitad de su ejecución hubo algún problema (por ejemplo, el producto pedido no está en stock, por lo tanto no se puede generar el envío).
- La transacción garantiza la atomicidad de la operación: o se hacen todas las operaciones, o no se hace ninguna.
- Trabajar con transacciones puede ser esencial para mantener la integridad de los datos.
  - Ej: se puede dar el caso de que se descuenta el stock de un producto antes de proceder a su envío, pero cuando se va a generar la cabecera del pedido, la aplicación cliente sufre un corte en las comunicaciones y no da tiempo a generarlo. Esto supone una pérdida de stock.

Transacciones (II)

- Un SGBD suele tener activado por defecto AUTOCOMMIT=ON → cada comando SQL ejecutado, será considerado como un transacción independiente.
- Para activar las transacciones de múltiples sentencias, hay que establecer el modo AUTOCOMMIT=OFF.
  - A partir de ese momento, todos los comandos SQL enviados al SGBD tendrán que terminarse con una orden COMMIT o una orden ROLLBACK.
  - De este modo, se asegura la integridad de los datos a un nivel más alto. Muchos SGBD requieren de una orden START TRANSACTION o START WORK para comenzar una transacción y otros lo hacen de forma implícita al establecer el modo autocommit=off.
- #MySQL, 3 formas para comenzar una transacción:
  - SET AUTOCOMMIT=0;
  - START TRANSACTION;
  - BEGIN WORK;
- Para terminar una transacción, tanto en MySQL como en ORACLE, hay que aceptar, o rechazar los cambios mediante:
  - COMMIT; #Acepta los cambios.
  - ROLLBACK; #Cancela los cambios

Transacciones (III)

- Ej:

```
SET AUTOCOMMIT=0;

#se actualiza el stock
UPDATE Productos SET Stock=Stock-2 WHERE CodigoProducto='AAAF102';

#se inserta la cabecera del pedido
INSERT INTO Pedidos VALUES (25,nowO,'Francisco Garcia','Pendiente de Entrega');

#se inserta el detalle del pedidoINSERT INTO DetallePedidos(CodigoPedido,CodigoProducto,Unidades) VALUES (25,'AAAF102',2);

#aceptar transacción
COMMIT;
```

Acceso concurrente a los datos (1)

- Cuando se utilizan transacciones, pueden ocurrir problemas de concurrencia en el acceso a los datos
  - Problemas ocasionados por el acceso al mismo dato por dos transacciones distintas.
- Tipos de problemas:
  - Dirty Read (Lectura Sucia): una transacción lee datos escritos por una transacción que no ha hecho COMMIT.
  - Nonrepeateable Read (Lectura No Repetible): una transacción vuelve a leer datos que leyó previamente y encuentra que han sido modificados por otra transacción.
  - Phantom Read (Lectura Fantasma): una transacción lee unos datos que no existían cuando se inició la transacción.

Acceso concurrente a los datos (II)

- El SGBD puede bloquear conjuntos de datos para evitar o permitir que sucedan estos problemas.
- Según el nivel de concurrencia que se desee, es posible solicitar al SGBD cuatro niveles de aislamiento:
  - **Read Uncommited** (Lectura no acometida): permite que sucedan los tres problemas. Las sentencias SELECT son efectuadas sin realizar bloqueos, por tanto, todos los cambios hechos por una transacción pueden verlos las otras transacciones.
  - **Read Commited** (Lectura acometida): los datos leídos por una transacción pueden ser modificados por otras transacciones. Se pueden dar los problemas de Phantom Read y Non Repeteable Read.
  - **Repeateable Read** (Lectura Repetible): tan solo se permite el problema del Phantom Read. Consiste en que ningún registro leído con un SELECT se puede cambiar en otra transacción.
  - **Serializable**: las transacciones ocurren de forma totalmente aislada a otras transacciones. Se bloquean las transacciones de tal manera que ocurren unas detrás de otras, sin capacidad de concurrencia. El SGBD las ejecuta concurrentemente si puede asegurar que no hay conflicto con el acceso a los datos.
- En MySQL, las tablas innodb tienen el nivel de aislamiento por defecto establecido en REPETEABLE READ. Se puede modificar:
  - Cambiando el fichero de configuración my.cnf
  - Ejecutando: SET TRANSACTION ISOLATION LEVEL {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE}

Manipulación y control de datos

1º DAW. Bases de datos. Colegio calasanz