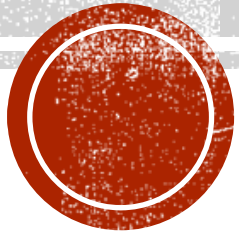
TEMA 6: TRATAMIENTO DE DATOS

IES PORTADA ALTA

BASE DE DATOS

1° DAM



ÍNDICE

- Inserción de registros
  - Cláusula INSERT
  - Cláusula REPLACE
  - Exportación/importación de datos
- Modificación de registros
- Borrado de registros
- Borrados/Modificaciones e integridad referencial
- Modificación de datos en vistas
- Transacciones
- Políticas de bloqueo de tablas
  - Comandos de bloqueo
  - Tipos de bloqueo
  - Adquisición/liberación de un bloqueo
  - Bloqueos y transacciones
  - Inserciones concurrentes

INTRODUCCIÓN

- En este tema vamos a empezar viendo la parte del lenguaje SQL relacionado con la edición de información y las operaciones que incluye, es decir, **inserción (INSERT), modificación (UPDATE) y eliminación (DELETE)** de los datos de nuestras bases de datos

CLAUSULA INSERT

- Es el tipo más simple de inserción. Admite tres tipos de sintaxis.

Formato 1

```
INSERT [LOW PRIORITY | DELAYED | HIGH PRIORITY] [IGNORE]
       [INTO] tbl name [PARTITION (partition name,...)]
       [(col name, ...)]
       {VALUES | VALUE} ({expr | DEFAULT}, ...),(...), ...
       [ON DUPLICATE KEY UPDATE
             col_name=expr
             [, col_name=expr] ... ]
INSERT [INTO] { nombre tabla | nombre vista } [ (lista columnas) ]
{ VALUES (expresion [, expresion]...) | sentencia_SELECT }
```

- Un ejemplo de INSERT sería:

- **INSERT INTO** prestamo (titulo, dni_usuario, fecha) **VALUE**
  ('Base de datos', '12345678Z', '2023-03-21');

CLAUSULA INSERT

- Algunas de las opciones son:
- ***LOW PRIORITY***: hace que la inserción se retrase mientras haya clientes leyendo de la tabla afectada. Sirve solo para tablas que admiten bloqueos, como son las ***MyISAM***, ***MEMORY*** y ***MERGE***.
- ***DELAYED***: permite continuar con otras operaciones mientras la inserción se retrasa hasta que no haya clientes accediendo a la tabla, es decir, es como la anterior, pero permite continuar trabajando.
- ***HIGH PRIORITY***: deshabilita el efecto de la variable de sistema ***-low-priority-updates*** si esta activa (=1). Esta variable hace que las operaciones de modificación tengan menor prioridad que las de consulta.
- ***IGNORE***: obvia los errores en la inserción. Por ejemplo, no podremos insertar registros con claves repetidas, pero tampoco nos informará del error.
- ***INTO***: es una cláusula opcional para indicar el nombre de la tabla o vista.

CLAUSULA INSERT

- ***PARTITION***: especifica las particiones donde se pretenden insertar los datos.
- ***DEFAULT***: sirve para usar el valor del campo por defecto creado cuando se definió la tabla.
- ***Lista*** columnas: podemos incluir entre paréntesis grupos de valores para insertar más de una fila en la misma sentencia.
- ***Values***: es el conjunto de valores expresados mediante expresiones (normalmente valores literales) o procedentes de una consulta.
- ***ON DUPLICATE KEY UPDATE***: cuando el valor de una clave (primaria o secundaria) se repite, esta cláusula permite actualizar uno o varios campos del registro correspondiente.

CLAUSULA INSERT

- Con SET

Formato 2

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
       [INTO] tbl name [PARTITION (partition name,...)]
       SET col_name = {expr I DEFAULT},
       [ ON DUPLICATE KEY UPDATE
             col_name=expr
                   [ , col name = expr] ... ]
```

- En este caso se permite especificar el nombre y valor de las columnas explícitamente con SET.
- Ejemplo: Para insertar un nuevo cliente de nombre 'Gabriel González' y el resto de valores por defecto:
- `INSERT INTO CLIENTE SET nombre='Gabriel', apellido1 = 'González';`

CLAUSULA INSERT

- Insertando desde SELECT

Formato 3

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
       [INTO] tbl name [PARTITION (partition name,...)]
       [(col_name,...)]
       SELECT ...
       [ ON DUPLICATE KEY UPDATE
             col_name=expr
             [, col name=expr]...]
```

- Ahora podemos insertar los valores procedentes de otra tabla o vista especificada mediante un SELECT.
- `INSERT INTO jugadores_historico SELECT* FROM jugadores WHERE YEAR(fecha alta) < YEAR(cur date());`

CLAUSULA REPLACE

- Funciona igual que INSERT excepto por el hecho de que si el valor de la clave primaria del registro a insertar coincide con un valor existente, éste se borrará para insertar el nuevo.
- La sintaxis tiene también tres formas.

Formato 1

```
REPLACE [LOW PRIORITY | DELAYED]
        [INTO] tbl name
        [PARTITION (partition name,... )]
        [(col_name,... )]
        {VALUES | VALUE} ({expr | DEFAULT},...),(...), ...
```

Formato 2

```
REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl name
        [PARTITION (partition_name,...)]
        SET col name = {expr | DEFAULT}, ...
```

Formato 3

```
REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl name
        [PARTITION (partition name,...)]
        [(col_name, ... )]
        SELECT ...
```

EXPORTACIÓN/IMPORTACIÓN DE DATOS LOAD DATA I

Permite cargar datos de un fichero de texto a tablas.

Sintaxis:

```
LOAD DATA [LOW PRIORITY | CONCURRENT][LOCAL] INFILE
'fichero'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    [IGNORE number LINES]
```

EXPORTACIÓN/IMPORTACIÓN DE DATOS LOAD DATA I

- Ejemplo:
- Disponemos de un archivo '**clientes.csv**' en nuestro ordenador, en la ruta: 'C:\datos\' de Windows. Para cargar los datos del fichero csv en una tabla '**cliente**' de una base de datos de MariaDB, pondremos:

```
LOAD DATA INFILE 'C:/datos/clientes.csv'
INTO TABLE cliente
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

Observar que para indicar la ruta se utiliza la barra / y no la barra invertida que usa Windows en su explorador.

EXPORTACIÓN/IMPORTACIÓN DE DATOS SOURCE

- Podemos ejecutar ficheros de comandos para ejecutar un conjunto de instrucciones INSERT del siguiente modo:

```
C:\> mysql –uusuario –ppassword < ruta_fichero_de_comandos
```

También disponemos del comando SOURCE ejecutable desde el cliente mysql:

```
mysql> source ruta_fichero_de_comandos
```

O equivalentemente con \.:

```
mysql> \. ruta_fichero_de_comandos
```

EXPORTACIÓN/IMPORTACIÓN DE DATOS
EXPORTAR DATOS

- Mediante la cláusula *SELECT*

```
SELECT…[INTO OUTFILE 'file_name'
export_options
    | INTO DUMPFILE 'file_name'
```

- Mediante *mysqldump*

```
mysqldump [opciones]
nombre_de_base_de_datos [tablas]
mysqldump [opciones] --databases DB1 [DB2 DB3 ...]
mysqldump [opciones] --all-databases
```

MODIFICACIÓN DE DATOS: UPDATE I

- **Formato para una sola tabla**

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET col_name1={expr1|DEFAULT}
      [,col_name2={expr2|DEFAULT}]...
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

- **Formato para varias tablas**

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
    SET col_name1={expr1|DEFAULT} [,
    col_name2={expr2|DEFAULT}] ...

     [WHERE where_condition]
```

MODIFICACIÓN DE DATOS: UPDATE II

Ejemplo actualización para una sola tabla

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
```

Sube el salario de los jugadores del equipo 5 en 1000 euros:

```
UPDATE jugadores SET salario=salario+1000 WHERE equipo=5;
```

Ejemplo actualización en varias tablas

Añade un campo id_capitan en la tabla equipo y actualiza los valores según la información de la tabla jugador:

```
UPDATE equipos e JOIN jugador j ON e.id_equipo=j.id_jugador SET
e.id_capitan=j.capitan;
```

BORRADO DE REGISTROS: DELETE I

- **Formato para una sola tabla**

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
[WHERE where_condition] [ORDER   BY ...]
[LIMIT row_count]
```

- **O usando TRUNCATE**

```
TRUNCATE TABLE tbl_name
```

BORRADO DE REGISTROS: DELETE II

- **Formato 1 para varias tablas**

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*]] ...
    FROM table_references
    [WHERE where_condition]
```

- **Formato 2 para varias tablas**

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name[.*] [, tbl_name[.*]] ...
    USING table_references
    [WHERE where_condition]
```

BORRADO DE REGISTROS: DELETE III

- **Ejemplo borrado varias tablas**

-Si disponemos de tres tablas t1, t2 y t3 podemos borrar datos de dos de ellas, t1 y t2, usando cualquiera de los siguientes comandos:

```
DELETE t1, t2 FROM t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;

DELETE FROM t1, t2 USING t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

- **Ejemplo borrado con cláusulas ORDER BY y LIMIT**

-Eliminar todos los equipos que no hayan jugado partidos como locales:

```
DELETE equipo FROM equipo e LEFT JOIN partido ON e.id_equipo = p.elocal
WHERE p.elocal IS NULL;
```

BORRADO/MODIFICACIÓN E INTEGRIDAD REFERENCIAL

- **Acciones posibles frente al borrado o modificación de datos con integridad referencial**
- **NO ACTION**

Se impide la eliminación o modificación de un registro en una tabla que tiene registros relacionados (mediante alguno de sus tributos comunes) en otras tablas.

- **RESTRICT**

Equivalente a NO ACTION.

- **SET NULL**

Permite borrar o actualizar un registro en la tabla 'padre' poniendo a NULL el campo o campos relacionados en la tabla 'hija' (salvo que se hayan definido como NOT NULL).

- **CASCADE**

Es el valor más habitual ya que propaga las modificaciones o borrados a los registros de la tabla hija relacionados con los de la tabla padre.

- **SET DEFAULT**: No es posible usar esta opción cuando estamos trabajando con InnoDB.

BORRADO/MODIFICACIÓN E INTEGRIDAD REFERENCIAL

- **Aspectos a considerar según la operación**

❖**INSERCION DE DATOS**

Genera errores en carga masiva de datos

❖**MODIFICACIÓN DE DATOS**

Los cambios se suelen propagar en cascada. Si está a SET NULL puede generar demasiados valores nulos

❖**ELIMINACIÓN DE DATOS**

Es peligrosa si está en modo CASCADE

TRANSACCIONES

Propiedades ACID

1. Atomicidad

Asegura que se realizan todas las operaciones o ninguna, no puede quedar a medias

2. Consistencia o integridad

Asegura que solo se empieza lo que se puede acabar

3. Aislamiento

Asegura que ninguna operación afecta a otras pudiendo causar errores

4. Durabilidad

Asegura que una vez realizada la operación, ésta no podrá cambiar y permanecerán los cambios

TRANSACCIONES II

Secuencia transacción en MySQL

1. Iniciar una transacción con el uso de la sentencia **START TRANSACTION** o **BEGIN**

2. Actualizar, insertar o eliminar registros en la base de datos

3. Si se quieren los cambios a la base de datos, completar la transacción con el uso de la sentencia **COMMIT**. Únicamente cuando se procesa un **COMMIT** los cambios hechos por las consultas serán permanentes

4. Si sucede algún problema, podemos hacer uso de la sentencia **ROLLBACK** para cancelar los cambios que han sido realizados por las consultas que han sido ejecutadas hasta el momento

TRANSACCIONES III

Ejemplo transacción en MySQL

1. Comprobamos el estado de la variable autocommit

```
SHOW VARIABLES LIKE 'autocommit';
```

   Si tiene el valor 1 la desactivamos con **SET.** Otra opción es realizar los ejemplos con **START TRANSACTION**

2. Creamos una tabla, en la base de datos test, del tipo InnoDB e insertamos algunos datos.

   Para crear una tabla **InnoDB**, procedemos con el código SQL estándar **CREATE TABLE**, pero debemos especificar que se trata de una tabla del tipo **InnoDB (TYPE= InnoDB).** La tabla se llamará **trantest** y tendrá un campo numérico. Primero activamos la base test con la instrucción **USE** para después crear la tabla e introducir algunos valores.

```
USE test;
CREATE TABLE trantest(campo INT NOT NULL PRIMARY KEY) TYPE = InnoDB;
INSERT INTO trantest VALUES(1),(2),(3);
```

TRANSACCIONES III

Ejemplo transacción en MySQL

3. Una vez cargada la tabla iniciamos una transacción.

```
BEGIN;
INSERT INTO trantest VALUES(4);
```

Si en este momento ejecutamos un **ROLLBACK**, la transacción no será completada y los cambios realizados sobre la tabla no tendrán efecto

```
mysql> ROLLBACK;
```

Si ahora hacemos un **SELECT** para mostrar los datos de **trantest** veremos que no

se ha llegado a producir ninguna inserción

TRANSACCIONES IV

4. Ahora vamos a ver qué sucede si perdemos la conexión al servidor antes de que la transacción sea completada

```
BEGIN;
INSERT INTO trantest VALUES(4);
SELECT * FROM trantest;
+-------+
| campo |
+-------+
| 1 |
| 2 |
| 3 |
| 4 |
4 rows in set (0.00 sec)
```

Cuando obtengamos de nuevo la conexión, podemos verificar que el registro no se insertó, ya que la transacción no fue completada

TRANSACCIONES V

5. Ahora vamos a repetir la sentencia INSERT ejecutada anteriormente, pero haremos un COMMIT antes de perder la conexión al servidor al salir del monitor de MySQL

```
BEGIN;
INSERT INTO innotest VALUES(4);
COMMIT;
EXIT;
Bye
```

Una vez que hacemos un COMMIT, la transacción es completada, y todas las sentencias SQL que han sido ejecutadas previamente afectan de manera permanente a las tablas de la base de datos.

TRANSACCIONES VI

Otros comandos transacciones

- **SAVEPOINT id**

Es una instrucción que permite nombrar cierto paso en un conjunto de instrucciones de una transacción

- **ROLLBACK TO SAVEPOINT id**

Permite deshacer el conjunto de operaciones realizadas a partir del identificador de savepoint

- **RELEASE SAVEPOINT id**

Elimina o libera el savepoint creado

Cualquier operación **COMMIT** o **ROLLBACK** sin argumentos eliminará todos los savepoints creados

COMANDOS BLOQUEO DE TABLAS

- **Bloqueo de todas las tablas**

```
LOCK TABLES WITH READ LOCK
```

- **Desbloqueo de tablas**

```
UNLOCK TABLES
```

- **Sintaxis General**

```
LOCK TABLES
tbl_name [[AS] alias] lock_type
[, tbl_name [[AS] alias] lock_type] ...
lock_type:
READ [LOCAL]
| [LOW_PRIORITY] WRITE
```

TIPOS DE BLOQUEO

✓**Bloqueo de lectura**

El cliente que obtiene el bloqueo puede leer pero ni él ni nadie puede modificar el registro bloqueado

```
READ [LOCAL]
```

✓**Bloqueo escritura**

Solo el usuario que obtiene el bloqueo puede leer y escribir sobre el registro bloqueado

```
[LOW PRIORITY] WRITE
```

ADQUISICIÓN/LIBERACIÓN DE UN BLOQUEO

Cuando hay varias solicitudes de bloqueo tiene prioridad el de escritura

Política de bloqueos seguida por MySQL

1. Ordena internamente las tablas a bloquear.
2. Si una tabla debe bloquearse para lectura y escritura sitúa la solicitud de bloqueo de escritura en primer lugar.
3. Se bloquea cada tabla hasta que la sesión obtiene todos sus bloqueos.

BLOQUEOS Y TRANSACCIONES I

Variable autocommit

- Con valor=1

Está activada lo que implica que cada comando SQL es en si mismo una transacción

El comando START TRANSACTION desactiva este comportamiento.

- Con valor =0

Se desactiva lo que hace que sólo los comandos **COMMIT** y **ROLLBACK** confirmen o revoquen una sentencia SQL

BLOQUEOS Y TRANSACCIONES II

Tipos de bloqueo en InnoDB

- **Bloqueo compartido: tipo s**

  Varias transacciones adquieren bloqueos sobre las mismas filas. Ninguna puede modificarlas hasta que se liberen todos los bloqueos

- **-Bloqueo exclusivo: tipo x**

  Permite adquirir bloqueos para modificación o borradoEn este caso las transacciones que deseen adquirir un bloqueo exclusivo deberán esperar a que se libere el bloqueo sobre las filas afectadas

- **-Intención de bloqueo: permite evitar conflictos (deadlocks) entre varias sesiones concurrentes**

  De lectura: IS

  De escritura: IX

BLOQUEOS Y TRANSACCIONES III

Ejemplo bloqueo IS

Supongamos que queremos agregar un registro en la tabla jugador pero asegurándonos de que existe su equipo (suponemos además que no hay integridad referencial).

Para asegurar que existe el equipo haríamos una consulta sobre equipo para comprobarlo y después insertaríamos el jugador. Sin embargo nadie nos asegura que entre medio se elimine dicho equipo. Para evitarlo haríamos la consulta de este modo:

```
SELECT * FROM equipo WHERE nombre='id_equipo' LOCK IN SHARE MODE;
```

Con lo que conseguimos un bloqueo de lectura de modo que mientras no se confirme o deshaga la transacción nadie podrá modificar los datos de equipo.

BLOQUEOS Y TRANSACCIONES IV

Ejemplo bloqueo IX

Si queremos añadir un nuevo jugador con un id determinado por el valor de un contador almacenado en la tabla contador. Si hacemos un bloqueo de lectura es posible que más de una transacción acceda al mismo valor de contador para dos jugadores distintos de manera que se producirá un error al ser un campo clave y no poder repetirse. Para evitarlo usaríamos un bloque de escritura o exclusivo para después incrementar el contador con la siguiente orden:

```
SELECT num_jugadores FROM contador FOR UPDATE;

UPDATE num_jugadores SET num_jugadores= num_jugadores+1;
```

Ahora cualquier transacción que quiera leer el campo num_jugadores deberá esperar a que se libere el bloqueo exclusivo y por tanto obtendrá el valor correcto.

BLOQUEOS Y TRANSACCIONES V

Niveles de aislamiento

1.READ UNCOMITTED

Las lecturas se realizan sin bloqueo de manera que podemos estar leyendo un dato que ya ha sido modificado por otra transacción. Es lo que se denomina lectura inconsistente o sucia.

2.READ COMMITED

En este tipo de lectura en cada instrucción se usan los valores más recientes y no los de comienzo del bloqueo. Es decir, si en alguna instrucción se modifica un valor que después se usará en otra, el valor será el modificado y no el original cuando se adquirió el bloqueo.

3. REPETEABLE READ

Es el valor por defecto. En este caso se obtiene el valor establecido al comienzo de la transacción. Es decir, aunque durante la transacción los valores cambien, se tendrán en cuenta siempre los del comienzo.

4. SERIALIZABLE

Es como la anterior con la diferencia de que ahora de manera interna se convierten los SELECT en SELECT… LOCK IN SHARE MODE si autocommit está desactivado.

BLOQUEOS Y TRANSACCIONES VI

- **Modificación niveles aislamiento**

Comando SET TRANSACTION

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL {
READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

BLOQUEOS Y TRANSACCIONES VII

Inserciones concurrentes

Permiten la lectura simultánea de datos

Comportamiento controlado por una variable de sistema llamada concurrent_insert cuyos valores posibles son:

AUTO ó 1	se activan las inserciones concurrentes
0	se impiden las inserciones concurrentes
2	se permiten las inserciones incluso aunque haya filas borradas entre los datos de la tabla