

# Bases de Datos

Gestión de Errores /  
Transacciones (Tx)/Cursores /  
Triggers



# Índice



- ***Gestión de Errores***
  - DECLARE HANDLER
- **Transacciones**
  - COMMIT / ROLLBACK
  - ACID
  - Puntos de Parada
- **Cursores**
  - Declare
  - Open
  - Fetch
  - Close
- **Triggers**

# Gestión de Errores



- ¿Qué sucede cuando un procedimiento tiene un error?
  - No encuentra una tabla, accedemos erróneamente a un campo que no existe, etc..
- MariaDB devolverá un error con un código:  
<https://mariadb.com/kb/en/mariadb-error-codes/>
  - 1046 bd no seleccionada, 1146 tabla no existente
- Solución: declarar un manejador de errores en nuestro código

# DECLARE HANDLER



- <https://mariadb.com/kb/en/declare-handler/>

```
DECLARE accionManejador HANDLER  
    FOR valorCondicion [, valorCondicion] ...  
    sentencia
```

```
accionManejador:  
    CONTINUE | EXIT
```

```
valorCondicion:  
    mariadb_error_code  
    | SQLSTATE [VALUE] sqlstate_value  
    | condition_name  
    | SQLWARNING  
    | NOT FOUND  
    | SQLEXCEPTION
```

# A tener en cuenta...



- El manejador se lanzará para una o más condiciones.
  - Al lanzarse, se puede ejecutar un única instrucción (por ejemplo `set x=5;`) o un bloque de instrucciones.
- Los manejadores se deben declarar después de las variables locales.

# Ejemplos manejadores



```
DECLARE CONTINUE HANDLER FOR 1051
BEGIN
  -- cuerpo del manejador
END;
-- Intenta acceder a una tabla que no existe
```

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
BEGIN
  - cuerpo del manejador
END;
-- Intenta acceder a una tabla que no existe
```


# Alias de error



- `SQLWARNING` : valores de `SQLSTATE` que empiezan por `01`.
- `NOT FOUND` : valores de `SQLSTATE` que empiezan por `02`.
  - Se utiliza con cursores cuando se ha llegado al final de los datos, lo que provoca una condición `NO DATA` con `SQLSTATE` igual a `02000`
- `SQLException` : valores de `SQLSTATE` que empiezan por `00`, `01` o `02`.

# Ejemplo



 11plsql-declare-continue-handler.sql

Raw

```
1  DROP DATABASE IF EXISTS pruebas;
2  CREATE DATABASE pruebas;
3  USE pruebas;
4
5  CREATE TABLE pruebas.t (s1 INT, PRIMARY KEY (s1));
6
7  DELIMITER //
8  CREATE PROCEDURE ejemploM()
9  BEGIN
10     -- El código 23000 se lanza al violar la restricción de integridad
11     DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x = 1;
12     SET @x = 1;
13     INSERT INTO pruebas.t VALUES (1);
14     SET @x = 2;
15     INSERT INTO pruebas.t VALUES (1);
16     SET @x = 3;
17 END
18 //
19 DELIMITER ;
```

- ¿Cuanto valdrá @x después de invocar a ejemploM?
- Si cambiamos el manejador de CONTINUE a EXIT, ¿Cuánto valdría @x?





# Transacciones. Definición

Una transacción SQL es un conjunto de sentencias SQL que se ejecutan formando una unidad lógica de trabajo ( *LUW Logic Unit Work*) es decir, en forma **atómica**.

Una transacción SQL finaliza con un `COMMIT` acepta todos los cambios o con `ROLLBACK` para deshacerlos.

MariaDB sólo permite realizar transacciones con el motor de almacenamiento InnoDB.

El uso de transacciones nos permite realizar operaciones de forma segura y recuperar datos si se produce algún fallo en el servidor durante la transacción, pero por otro lado las transacciones pueden aumentar el tiempo de ejecución de las instrucciones.

# ACID



Las transacciones deben cumplir cuatro propiedades para garantizar de se puedan realizar de forma segura.

- **Atomicidad:** La transacción es indivisible
  - O se ejecutan todas la sentencias o ninguna.
- **Consistencia:** Después de una transacción la BD estará en un estado válido y consistente.
- **Aislamiento:** Cada transacción está aislada del resto de transacciones y el acceso a los datos se hace de forma exclusiva.
  - Si una transacción quiere acceder de forma concurrente a los datos que están siendo utilizados por otra transacción, no podrá hacerlo hasta que la primera haya terminado.
- **Durabilidad:** Los cambios que realiza una transacción sobre la base de datos son permanentes.

# AUTOCOMMIT



- Se aceptan automáticamente todos los cambios realizados y no es posible deshacerlos. Por defecto está activada.
- Consultar el autocommit
  - `SELECT @@AUTOCOMMIT;`
- Desactivar el autocommit
  - `SET AUTOCOMMIT = 0;`
  - Al desactivarlo tendremos una transacción abierta. Los cambios sólo se aplicarían ejecutando la sentencia `COMMIT` de forma explícita.
- Activar el autocommit
  - `SET AUTOCOMMIT = 1;`

# Pasos



- 1) Indicar que vamos a realizar una transacción con la sentencia `START TRANSACTION`, `BEGIN` o `BEGIN WORK`.
- 2) Realizar las operaciones de manipulación de datos sobre la base datos (insertar, actualizar o borrar filas).
- 3) Dos posibilidades:
  - 1) Si las operaciones se han completado y queremos que los cambios se apliquen de forma permanente → `COMMIT`.
  - 2) Si durante las operaciones ocurre algún error y no queremos aplicar los cambios realizados, podemos deshacerlos → `ROLLBACK`.

# START TRANSACTION



- <https://mariadb.com/kb/es/start-transaction/>

```
START TRANSACTION
```

```
    [caracteristicasTx [, caracteristicasTx] ...]  
| BEGIN [work]
```

```
COMMIT
```

```
ROLLBACK
```

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

```
caracteristicasTx: {  
    WITH CONSISTENT SNAPSHOT |  
    READ WRITE |  
    READ ONLY  
}
```

# Ejemplo Cliente



`<>` 11plsql-start-transaction.sql

Raw

```
1 DROP DATABASE IF EXISTS pruebas;
2 CREATE DATABASE pruebas CHARACTER SET utf8mb4;
3 USE pruebas;
4
5 CREATE TABLE cliente (
6     id INT UNSIGNED PRIMARY KEY,
7     nombre VARCHAR (32)
8 );
9
10 START TRANSACTION;
11 INSERT INTO cliente VALUES (1, 'Pedro');
12 COMMIT;
13 SET AUTOCOMMIT=0;
14 INSERT INTO cliente VALUES (2, 'Laura');
15 INSERT INTO cliente VALUES (33, 'Fran');
16 DELETE FROM cliente WHERE nombre = 'Fran';
```

- ¿Qué contiene la tabla cliente?
- Si ejecutamos ROLLBACK, ¿Qué contendrá ahora cliente?

# Ejemplo Cuentas



11psql-start-transaction-errores.sql

Raw

```
1 CREATE TABLE cuentas (  
2     id INTEGER UNSIGNED PRIMARY KEY,  
3     saldo DECIMAL(11,2) CHECK (saldo >= 0)  
4 );  
5  
6 INSERT INTO cuentas VALUES (1, 1000);  
7 INSERT INTO cuentas VALUES (2, 2000);  
8 INSERT INTO cuentas VALUES (3, 0);  
9  
10 START TRANSACTION;  
11 UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1;  
12 UPDATE cuentas SET saldo = saldo + 100 WHERE id = 2;  
13 COMMIT;  
14  
15 -- ¿Qué contiene cuentas en este momento?
```

```
17 -- ¿Qué sucede cuando una cuenta no existe?  
18  
19 START TRANSACTION;  
20 UPDATE cuentas SET saldo = saldo - 100 WHERE id = 9999;  
21 UPDATE cuentas SET saldo = saldo + 100 WHERE id = 2;  
22 COMMIT;  
23  
24 -- ¿Y si una cuenta no tiene saldo?  
25  
26 START TRANSACTION;  
27 UPDATE cuentas SET saldo = saldo - 100 WHERE id = 3;  
28 UPDATE cuentas SET saldo = saldo + 100 WHERE id = 2;  
29 COMMIT;
```

# Puntos de Parada




- Posicionan paradas entre los pasos de una transacción.
- <https://mariadb.com/kb/en/savepoint/>
- `SAVEPOINT etiqueta`: Establece un punto de parada dentro de la transacción, utilizando una etiqueta.
- `ROLLBACK TO [SAVEPOINT] etiqueta`: Hace un ROLLBACK deshaciendo sólo las instrucciones que se han ejecutado hasta el punto de parada indicado.
- `RELEASE SAVEPOINT etiqueta`: Elimina un punto de parada.



# Ejemplo



 11plsql-savepoint.sql

Raw

```
1  CREATE TABLE producto (  
2      id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,  
3      nombre VARCHAR(100) NOT NULL,  
4      precio DOUBLE  
5  );  
6  
7  INSERT INTO producto (id, nombre) VALUES (1, 'Primero');  
8  INSERT INTO producto (id, nombre) VALUES (2, 'Segundo');  
9  INSERT INTO producto (id, nombre) VALUES (3, 'Tercero');  
10  
11  START TRANSACTION;  
12  INSERT INTO producto (id, nombre) VALUES (4, 'Cuarto');  
13  SAVEPOINT sp1;  
14  INSERT INTO producto (id, nombre) VALUES (5, 'Quinto');  
15  INSERT INTO producto (id, nombre) VALUES (6, 'Sexto');  
16  ROLLBACK TO sp1;
```

# Transacciones en Procedimientos



- Se utiliza un manejador para hacer **ROLLBACK**

```
CREATE PROCEDURE transaccionEnMariaDB()  
BEGIN  
    DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING  
    BEGIN  
        -- ERROR, WARNING  
        ROLLBACK;  
    END;  
    START TRANSACTION;  
    -- Sentencias SQL  
    COMMIT;  
END
```

# Cursores



- Almacenan un conjunto de filas de una tabla en una estructura de datos que podemos ir recorriendo de forma secuencial.
- Sólo se pueden utilizar con sentencias SELECT.
- Propiedades:
  - Sólo lectura: No permiten actualizar los datos.
  - *Nonscrollable*: sólo pueden ser recorridos en una dirección y no podemos saltarnos filas.

# DECLARE CURSOR



- <https://mariadb.com/kb/en/declare-cursor/>

**DECLARE** nombreCursor **CURSOR FOR** sentenciaSQL

- Ejemplo:

```
declare curClientes cursor for select  
codigo_cliente, nombre_cliente from cliente
```

# OPEN y FETCH



- Una vez declarado, el cursor se abre y se recorre fila a fila hasta llegar al final.
- Se abre con **OPEN** `nombreCursor`
  - Reserva memoria y ejecuta la consulta, colocando el puntero en la primera fila
- Se recorre con **FETCH** `nombreCursor` **INTO** `listaVariables`.
  - Almacena el contenido de la fila a la que apunta el puntero
- Se comprueba el final dentro de un bloque `DECLARE HANDLER FOR` **NOT FOUND**



# *Trigger* (disparador)

- Objeto de la base de datos que está asociado con una tabla y que se activa cuando ocurre un evento sobre la tabla.
- Posibles eventos:
  - **INSERT**: El *trigger* se activa cuando se inserta una nueva fila sobre la tabla asociada.
  - **UPDATE**: Se activa cuando se actualiza una fila.
  - **DELETE**: Se activa cuando se elimina una fila.
- Dentro del disparador, **NEW** referencia al nuevo registro, y **OLD** al antiguo.

# CREATE TRIGGER



- <https://mariadb.com/kb/en/create-trigger/>

```
CREATE [OR REPLACE] TRIGGER nombreTrigger
cuandoTrigger eventoTrigger
ON nombreTabla FOR EACH ROW
[ordenTrigger]
cuerpo
```

```
cuandoTrigger: { BEFORE | AFTER }
```

```
eventoTrigger: { INSERT | UPDATE | DELETE }
```

```
ordenTrigger: { FOLLOWS | PRECEDES } otroNombreTrigger
```

# Ejemplo Triggers



11plsql-triggers.sql

Raw

```
1 CREATE TABLE alumnos (  
2     id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,  
3     nombre VARCHAR(50) NOT NULL,  
4     apellido1 VARCHAR(50) NOT NULL,  
5     apellido2 VARCHAR(50),  
6     nota FLOAT  
7 );  
8  
9 DELIMITER //  
10 DROP TRIGGER IF EXISTS triggerCheckNotaBeforeInsert //  
11 CREATE TRIGGER triggerCheckNotaBeforeInsert  
12     BEFORE INSERT ON alumnos FOR EACH ROW  
13 BEGIN  
14     -- NEW referencia al nuevo registro  
15     IF NEW.nota < 0 THEN  
16         set NEW.nota = 0;  
17     ELSEIF NEW.nota > 10 THEN  
18         set NEW.nota = 10;  
19     END IF;  
20 END  
21 //
```

```
23 DROP TRIGGER IF EXISTS triggerCheckNotaBeforeUpdate //  
24 CREATE TRIGGER triggerCheckNotaBeforeUpdate  
25     BEFORE UPDATE ON alumnos FOR EACH ROW  
26 BEGIN  
27     IF NEW.nota < 0 THEN  
28         set NEW.nota = 0;  
29     ELSEIF NEW.nota > 10 THEN  
30         set NEW.nota = 10;  
31     END IF;  
32 END  
33 //  
34  
35 DELIMITER ;  
36 INSERT INTO alumnos VALUES (1, 'Ana', 'Sánchez', 'Pérez', -1);  
37 INSERT INTO alumnos VALUES (2, 'Bruno', 'García', 'Morales', 12);  
38 INSERT INTO alumnos VALUES (3, 'Carlos', 'Serrano', 'López', 8.5);  
39  
40 SELECT * FROM alumnos;  
41  
42 UPDATE alumnos SET nota = -4 WHERE id = 3;  
43 UPDATE alumnos SET nota = 14 WHERE id = 3;  
44 UPDATE alumnos SET nota = 9.5 WHERE id = 3;  
45  
46 SELECT * FROM alumnos;
```



# Comandos



- `DROP TRIGGER nombreTrigger`
  - <https://mariadb.com/kb/en/drop-trigger/>
- `SHOW TRIGGERS`
  - <https://mariadb.com/kb/en/show-triggers/>
- `SHOW CREATE TRIGGER`
  - <https://mariadb.com/kb/en/show-create-trigger/>

# Referencias



- Apuntes José Juan Sánchez Hernández - IES Celia Viñas (Almería):  
<https://josejuansanchez.org/bd/unidad-11-teoria/index.html>  
<https://josejuansanchez.org/bd/unidad-12-teoria/index.html>
- Documentación MaríaDB:  
<https://mariadb.com/kb/en/programming-customizing-mariadb/>



# ¿Alguna pregunta?