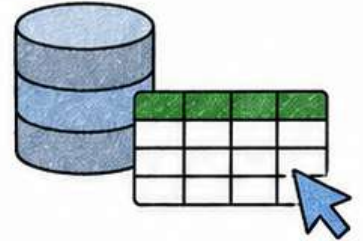


# Chapter 22: MySQL Cursors & Error Handling

## 22.1 Introduction to Cursors

- \* A **cursor** is a database object that allows you to process **individual rows** of the result set returned by a **SELECT** statement.
- \* In MySQL, cursors are used inside **stored procedures** to handle multiple rows one at a time.
- \* Cursors are very useful when we need to perform **complex processing** on each row.
- \* MySQL provides a way to **declare**, **open**, **fetch** data from, and **close** a cursor.



### Key Points :

- ✓ Cursor helps to iterate through the rows of a result set.
- ✓ Cursors are used only in stored procedures.
- ✓ Cursors process one row at a time.



## 22.2 DECLARE CURSOR

- \* Before using a cursor, it must be **declared**.
- \* The **DECLARE CURSOR** statement defines the result set for the cursor.
- \* It associates a cursor name with a **SELECT** query.



### Syntax :

```
DECLARE cursor_name CURSOR FOR
    SELECT column1, column2, ... FROM table_name
    WHERE condition;
```

### Example :

```
DELIMITER $$
CREATE PROCEDURE get_employees()
BEGIN
    DECLARE emp_cursor CURSOR FOR
        SELECT emp_id, emp_name, salary FROM employees;
END $$
DELIMITER ;
```



### Key Takeaway :



**DECLARE CURSOR** is the first step in using a cursor. It defines the result set that the cursor will work with.

## 22.3 OPEN

- \* After declaring a cursor, the next step is to **OPEN** it.
- \* The **OPEN** statement prepares the cursor to process the **result set**.
- \* It allocates the resources needed to execute the query associated with the cursor.

Syntax:

```
OPEN cursor_name ;
```

Example:

```
DELIMITER $$
CREATE PROCEDURE get_employees()
BEGIN
    DECLARE emp_cursor CURSOR FOR
        SELECT emp_id, emp_name, salary FROM employees;
    OPEN emp_cursor;
END$$
DELIMITER ;
```

Key Takeaway:



**OPEN** activates the cursor and prepares it to start fetching rows.



## 22.4 FETCH

- \* The **FETCH** statement is used to retrieve rows from the result set, one row at a time.
- \* Each time **FETCH** is executed, the next row is retrieved into the specified variables.
- \* It must be used after **OPEN** and before **CLOSE**.

Syntax:

```
FETCH cursor_name INTO variable1, variable2, ... ;
```

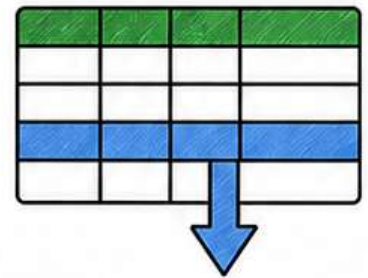
Example:

```
DELIMITER $$
CREATE PROCEDURE get_employees()
BEGIN
    DECLARE emp_id INT;
    DECLARE emp_name VARCHAR(100);
    DECLARE emp_salary DECIMAL(10,2);
    DECLARE emp_cursor CURSOR FOR
        SELECT emp_id, emp_name, salary FROM employees;
    OPEN emp_cursor;
    FETCH emp_cursor INTO emp_id, emp_name, emp_salary;
END$$
DELIMITER ;
```

Key Takeaway:



**FETCH** gets the next row from the cursor's result set and stores it in the specified variables.



## 22.5 CLOSE

- \* After fetching all rows, the cursor should be **closed**.
- \* The **CLOSE** statement releases the resources allocated to the cursor.
- \* Once a cursor is closed, it cannot be used again unless it is opened.

Syntax:

```
CLOSE cursor_name ;
```



Example:

```
DELIMITER $$
CREATE PROCEDURE get_employees()
BEGIN
    DECLARE emp_id INT ;
    DECLARE emp_name VARCHAR(100);
    DECLARE emp_salary DECIMAL(10,2);
    DECLARE emp_cursor CURSOR FOR
        SELECT emp_id, emp_name, salary FROM employees ;
    OPEN emp_cursor ;
    FETCH emp_cursor INTO emp_id, emp_name, emp_salary ;
    CLOSE emp_cursor ;
END$$
DELIMITER ;
```



Key Takeaway:



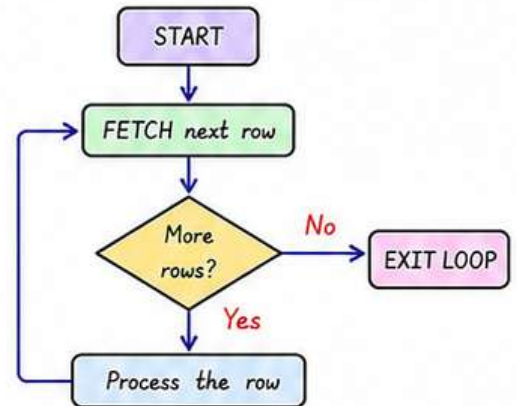
**CLOSE** frees the resources used by the cursor and marks the end of cursor usage.

## 22.6 Cursor Loops

- \* Cursors are usually used inside loops to process all rows one by one.
- \* A loop continues fetching rows until no more rows are available.
- \* We use a **handler** to detect when there are no more rows.
- \* When no more rows are found, the loop is exited.

Example:

```
DELIMITER $$
CREATE PROCEDURE get_employee_list()
BEGIN
    DECLARE done INT DEFAULT 0 ;
    DECLARE v_id INT ;
    DECLARE v_name VARCHAR(100);
    DECLARE v_salary DECIMAL(10,2);
    DECLARE emp_cursor CURSOR FOR SELECT emp_id, emp_name, salary FROM employees ;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1 ;
    OPEN emp_cursor ;
read_loop: LOOP
    FETCH emp_cursor INTO v_id, v_name, v_salary ;
    IF done = 1 THEN
        LEAVE read_loop ;
    END IF ;
    -- Process the row (e.g., display or do some action)
END LOOP ;
CLOSE emp_cursor ;
END$$
DELIMITER ;
```



Key Takeaway:



Cursor loops allow you to **fetch** and **process** each row one at a time until all rows have been handled.

|  |  |  |  |
|--|--|--|--|
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## 22.7 DECLARE HANDLER

- ★ The **DECLARE HANDLER** statement is used to define how MySQL should respond to **errors** or **warnings**.
- ★ It allows you to handle **errors** gracefully without terminating the program.
- ★ **Handlers** are useful in stored procedures, functions and triggers.
- ★ You must declare handlers **before** using them in the block.

### Syntax :

```
DECLARE handler_name HANDLER FOR condition_value  
statement ;
```

### Example :

```
DELIMITER $$  
CREATE PROCEDURE divide_numbers(IN a INT, IN b INT)  
BEGIN  
    DECLARE msg VARCHAR(100);  
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION  
        SET msg = 'An error occurred while dividing.' ;  
    SET @result = a / b ;  
    SELECT @result , msg ;  
END$$  
DELIMITER ;
```

### Key Takeaway :



**DECLARE HANDLER** helps control the flow of the program when **errors** or **warnings** occur.



## 22.8 SQLSTATE

- ★ **SQLSTATE** is a 5-character alphanumeric code that identifies errors, warnings and other conditions.
- ★ It provides a standard way to identify the type of exception.
- ★ It is used with **DECLARE HANDLER** to specify which errors to handle.

### Common SQLSTATE Classes :

| Class Code | Meaning                               | Description                                   |
|------------|---------------------------------------|-----------------------------------------------|
| 00         | Successful Completion                 | The statement was executed successfully.      |
| 01         | Warning                               | A warning occurred.                           |
| 02         | No Data                               | No data was found.                            |
| 22         | Data Exception                        | Invalid data value, number out of range, etc. |
| 23         | Integrity Constraint Violation        | Violation of integrity constraints.           |
| 42         | Syntax Error or Access Rule Violation | Syntax error or access violation.             |

### Key Takeaway :



**SQLSTATE** codes help identify and handle specific errors in a consistent and standardized way.

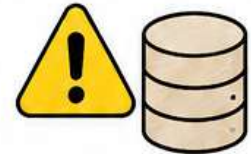


## 22.9 Error Handling in Stored Programs

- \* Error handling in stored programs ensures that the program **does not terminate** unexpectedly when an error occurs.
- \* We use **DECLARE HANDLER** along with **SQLSTATE** or MySQL error codes to handle errors.
- \* When an error occurs, the defined **handler** runs and control returns to the next statement after the block.
- \* This makes the program more **robust** and **user-friendly**.

### Syntax :

```
DECLARE handler_name HANDLER FOR condition_value  
statement ;
```

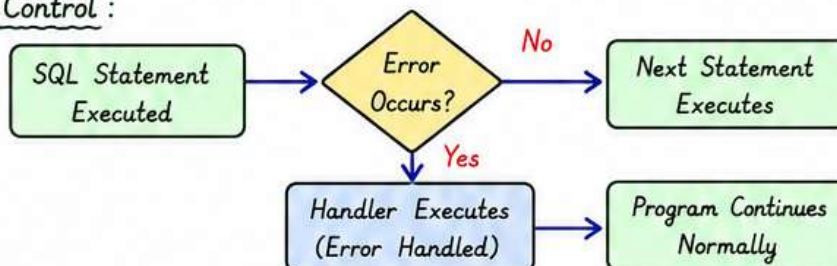


### Example :

```
DELIMITER $$  
CREATE PROCEDURE transfer_amount(IN from_ac INT, IN to_ac INT, IN amt DECIMAL(10,2))  
BEGIN  
    DECLARE done INT DEFAULT 0 ;  
    DECLARE CONTINUE HANDLER FOR 1062  
        SET done = 1 ; -- Duplicate entry error  
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION  
        SET done = 2 ; -- Any SQL exception  
    -- Program logic  
    UPDATE accounts SET balance = balance - amt WHERE account_id = from_ac ;  
    UPDATE accounts SET balance = balance + amt WHERE account_id = to_ac ;  
END$$  
DELIMITER ;
```



### Flow of Control :



### Common MySQL Error Codes :

| Error Code | Meaning                          | Description                                       |
|------------|----------------------------------|---------------------------------------------------|
| 1062       | Duplicate Entry                  | Duplicate key value violates a unique constraint. |
| 1146       | Table Doesn't Exist              | The table referenced does not exist.              |
| 1054       | Unknown Column                   | Column name is unknown.                           |
| 1452       | Foreign Key Constraint Fails     | Cannot add or update a child row.                 |
| 1364       | Field Doesn't Have Default Value | A required field is missing a value.              |



### Key Takeaway :



Error handling using **DECLARE HANDLER** and **SQLSTATE** makes stored programs reliable and prevents unexpected termination when errors occur.

