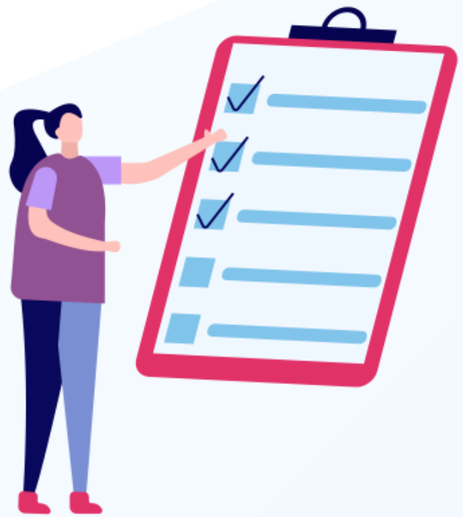
Stored Procedures and Triggers in SQL

Learning Objectives

By the end of this lesson, you will be able to:

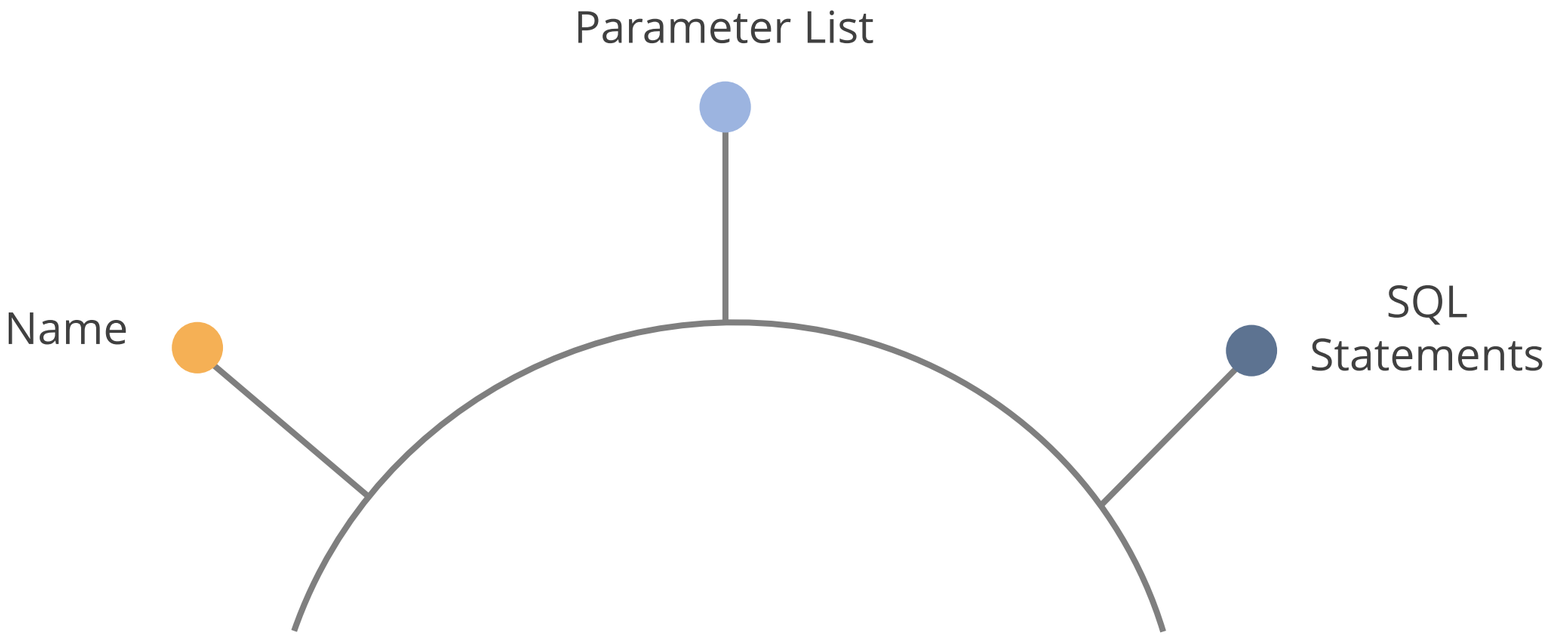
- List the advantages of stored procedures
- Interpret the various aspects of stored procedures
- Outline the different types of error handlers
- Analyze SQL triggers



Introduction to Stored Procedures

Stored Procedures

A stored procedure is a collection of precompiled SQL commands in a database.



When a procedure calls itself, then it is called a recursive stored procedure.

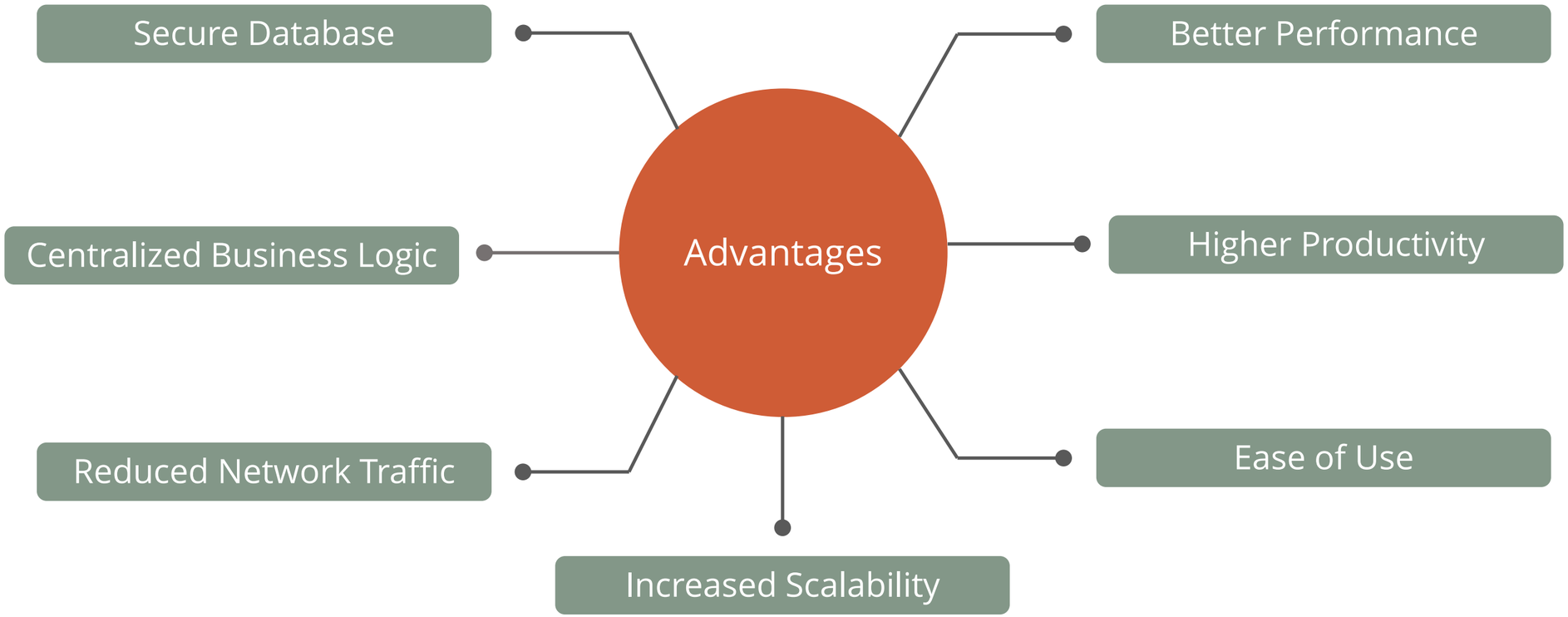
Stored Procedures

The explanation of each parameter in the stored procedure syntax is given below:

Procedure Name	• Refers to the name of the stored procedure
Declaration Section	• Refers to the declarations of all variables
Executable Section	• Refers to the section of code that is executable

Advantages of Stored Procedures

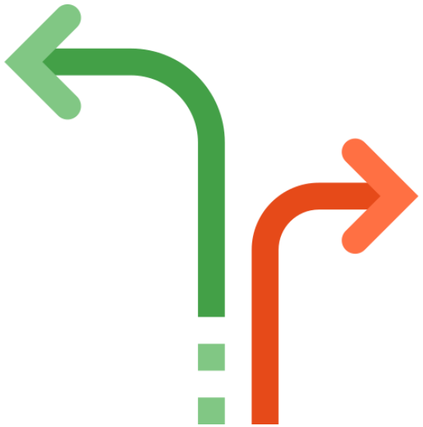
Advantages of Stored Procedures



Working with Stored Procedures

Changing the Default Delimiter

MySQL Workbench uses the delimiter **(;)** to separate statements and execute each statement distinctly.



- If MySQL Workbench is used to define a stored procedure with semicolon characters, then it considers the whole stored procedure as multiple statements.
- To tackle this, redefine the delimiter temporarily. This will pass the whole stored procedure to the MySQL workbench as a single statement.
- The command to redefine delimiter is **DELIMITER delimiter_character**.

Note: Backslash (\) is an escape character, and it must be avoided in MySQL.

Creating Stored Procedures

CREATE PROCEDURE keywords are used to create stored procedures.

SYNTAX

```
CREATE PROCEDURE procedure_name(parameter_list)
BEGIN
   statements;
END
```

In MySQL, code is written between the BEGIN and END keywords. The delimiter character is placed after END to conclude the procedure statement.

Executing Stored Procedures

To execute the stored procedure, you can use the following syntax with the CALL keyword:

SYNTAX

```
CALL [Procedure Name] ([Parameters])
```

If the procedure has parameters, then the parameter values must be specified in the parenthesis.

Removing Stored Procedures

DROP PROCEDURE statement is used to delete stored procedures.

SYNTAX

```
DROP PROCEDURE [IF EXISTS] stored_procedure_name;
```

- If you drop a procedure that does not exist without using the IF EXISTS option, MySQL shows an error.
- If you use the IF EXISTS option for the same condition, then MySQL shows a warning.

Stored Procedures: Example

Problem Statement: You are a junior DB administrator in your company. Your manager has asked you to retrieve data on employees with more than five years of experience, using a single command.

Objective: Use a stored procedure to retrieve the required data anytime.

Stored Procedures: Example

Step 1: You have a table on employees with details, such as employee ID, first name, last name, gender, role name, department, experience, country, and continent.

Emp_ID	Emp_Name	Role_name	Dept	Experience
260	Roy	Senior Data Scientist	Retail	7
620	Katrina	Junior Data Scientist	Retail	2
430	Steve	Associate Data Scientist	Finance	4
160	William	Lead Data Scientist	Automotive	12
52	Diana	Senior Data Scientist	Healthcare	6
366	Clair	Associate Data Scientist	Automotive	3
403	John	Lead Data Scientist	Finance	10

Stored Procedures: Example

Step 2: Create a stored procedure that displays the employees with more than five years of experience using the following command.

QUERY

```
DELIMITER &&
CREATE PROCEDURE get_mid_experience()
BEGIN
SELECT * FROM Emp_Table WHERE experience > 5;
END &&
```

Stored Procedures: Example

Step 3: Call for the stored procedure to return the results based on the specified condition.

QUERY

```
CALL get_mid_experience();
```

Output:

Emp_ID	Emp_Name	Role_name	Dept	Experience
260	Roy	Senior Data Scientist	Retail	7
160	William	Lead Data Scientist	Automotive	12
52	Diana	Senior Data Scientist	Healthcare	6
403	John	Lead Data Scientist	Finance	10

Using Variables in Stored Procedures

Variable is a named data object whose value can be changed during stored procedure execution.

They are used to store immediate results and are local to the stored procedure.

Declaring and Assigning Variables

DECLARE and SET keywords are used to declare and set variables.

Declaring Variables

```
DECLARE variable_name
datatype(size) [DEFAULT
    default_value];
```

Assigning Variables

```
SET variable_name = value;
```

Declaring and Assigning Variables: Example

Problem statement: You are a junior DB administrator, and your manager has asked you to identify the total number of employees in the employee table created earlier.

Objectives: Use the stored procedure to view the number of employees anytime and also declare a variable for total employees.

Declaring and Assigning Variables: Example

Step 1: Using the same employee table as earlier, create a stored procedure and declare a default variable 0.

QUERY

```
DELIMITER &&
CREATE PROCEDURE get_total_employees()
BEGIN
DECLARE totalemployee INT DEFAULT  0;
SELECT COUNT (*)
INTO totalemployee
FROM Emp_Table;
SELECT totalemployee;
END &&
DELIMITER ;
```

Declaring and Assigning Variables: Example

Step 2: Use the CALL function to view the stored procedure results.

QUERY

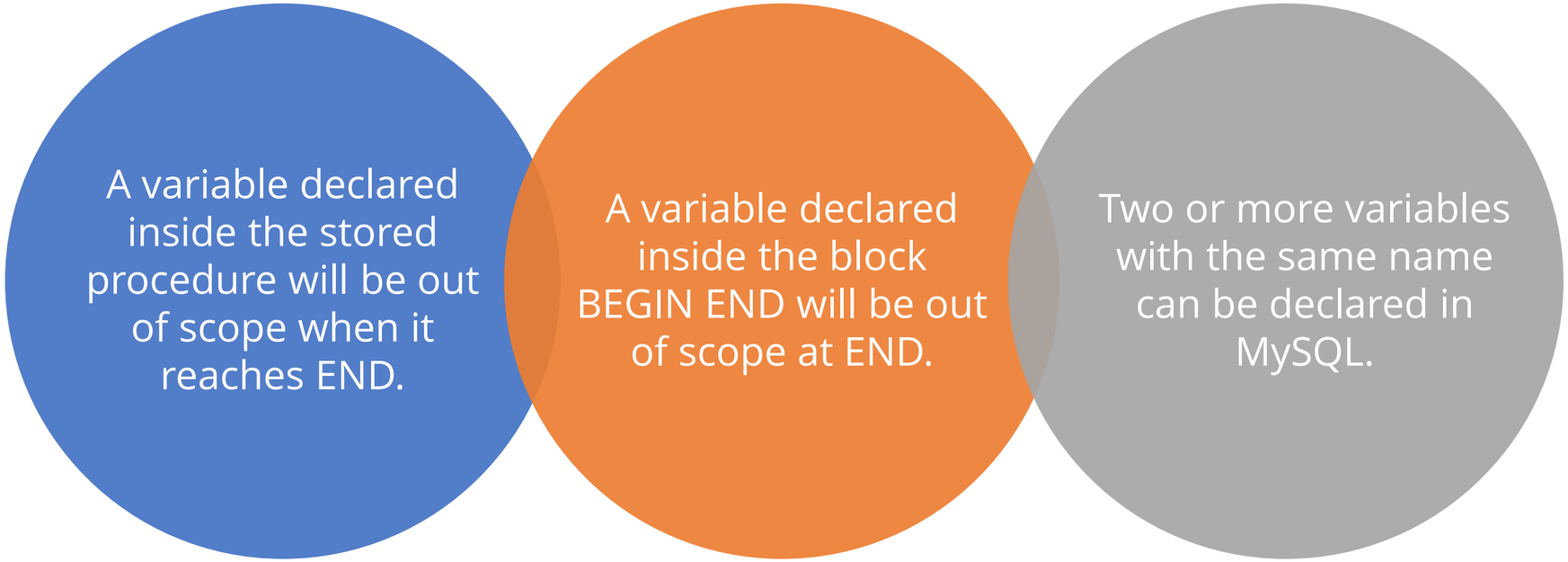
```
CALL get_total_employees()
```

Output:

	totalemployee
▶	7

Scope of Variables

Scope of a variable refers to the lifetime of a variable.



Note: A variable that begins with @ is called a session variable. It is accessible until the session ends.

Stored Procedures That Return Multiple Values

Parameter refers to a placeholder for a variable to store a value of a data type.

Generally, MySQL stored functions return a single value. To obtain multiple values, use stored procedures with INOUT or OUT parameters.

Stored Procedures That Return Multiple Values (IN)

This is the default mode, and parameter is the input here. The calling program must pass an argument to the stored procedure when it is defined.

- Values are protected in an IN parameter.
- When the values in the parameter are changed, the original value remains unchanged after the stored procedure ends.

Stored Procedures That Return Multiple Values: Example (IN)

Problem Statement: You are a junior DB administrator in your organization. Your manager has asked you to list the employee names in the automotive department.

Objective: Create a stored procedure with an IN parameter to extract employee names by specifying the department name.

Stored Procedures That Return Multiple Values: Example (IN)

Step 1: Create a procedure named employee of auto. Keep the department as the IN parameter.

QUERY

```
CREATE PROCEDURE Employee_of_Auto(
    IN Department VARCHAR(255)
)
BEGIN
    SELECT  Emp_Name, Dept
    FROM Emp_Table
    WHERE Dept = “Automotive”;
END
```

Stored Procedures That Return Multiple Values: Example (IN)

Step 2: Call the procedure with the mentioned department.

QUERY

```
CALL Emp_of_Auto("Automotive");
```

Output:

	Emp_Name	Dept
▶	William	Automotive
	Clair	Automotive

Stored Procedures That Return Multiple Values (OUT)

This parameter is used to pass a parameter as an output. Its value can be changed inside the stored procedure.

The initial value of the OUT parameter cannot be accessed by the stored procedure when it starts.

Stored Procedures That Return Multiple Values: Example (OUT)

Problem Statement: You are a junior DB administrator in your organization. Your manager has asked you to count the employees in the retail department.

Objective: Create a stored procedure with an OUT parameter and extract the required result.

Stored Procedures That Return Multiple Values: Example (OUT)

Step 1: Create a procedure called *employee count in retail* with the OUT parameter to count the employees in the retail department.

QUERY

```
DELIMITER &&
CREATE PROCEDURE Emp_Count_in_Retail ( OUT total_Emp INT)
BEGIN
SELECT count(Emp_ID) INTO total_Emp FROM Emp_Table WHERE Dept
= "Retail";
END &&
```

Stored Procedures That Return Multiple Values: Example (OUT)

Step 2: Call the created procedure to store the returned value. Pass a session variable named @Retail_Employees. Select values from these in a separate value called employee retail.

QUERY

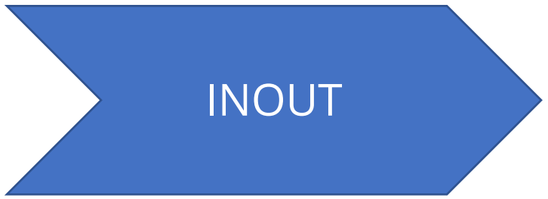
```
CALL Emp_Count_in_Retail (@Retail_Employees);
Select @Retail_Employees AS EmployeeRetail;
```

Output:

	EmployeeRetail
▶	2

Stored Procedures That Return Multiple Values (IN OUT)

This is a combination of IN and OUT parameters.



This specifies that the calling program can pass the argument and the stored procedure can modify the INOUT parameter.

Stored Procedures That Return Multiple Values (IN OUT)

Problem Statement: You are a junior DB administrator in your organization. Your manager wants to track total number of changes made each time when there is a new addition to an existing database.

Objective: Create a stored procedure with an IN OUT parameter to display the required count.

Stored Procedures That Return Multiple Values (IN OUT)

Step 1: Create a procedure iteration with count as the IN OUT parameter and increment as the IN parameter.

QUERY

```
CREATE PROCEDURE
Iterations (INOUT count int, IN increment int)
BEGIN
SET count = count + increment;
END
```

Stored Procedures That Return Multiple Values (IN OUT)

Step 2: Set Iterations to zero. Call the procedure with the variable when there is one change; repeat the process when there are five changes.

QUERY

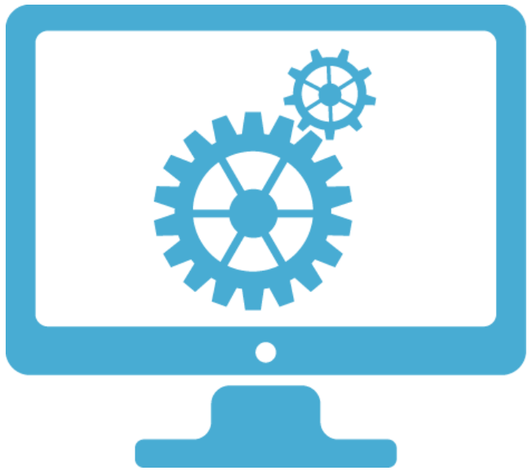
```
SET @Iterations = 0;
CALL Increment_Counter (@Iterations, 1);
select @Iterations;
CALL Increment_Counter (@Iterations, 5);
select @Iterations;
```

Output:

	@Iterations
▶	6

Stored Procedures with One and Multiple Parameters

Stored procedures can have one or more parameters, and these parameters are separated by commas.



Stored Procedures with One Parameter: Example

Problem Statement: You are a junior DB administrator in your organization. Your manager wants to identify an employee's experience based on just the employee ID and decide whether to give them a hike or not.

Objective: Create a stored procedure with employee ID as the parameter.

Stored Procedures with One Parameter: Example

Step 1: Create a stored procedure with the relevant employee details and keep employee ID as the parameter.

QUERY

```
DELIMITER $$
CREATE PROCEDURE GetEmpExp(eid int)
  BEGIN
      SELECT Emp_ID, Emp_Name
      Role_name, Dept, Experience
      FROM Emp_Table
      WHERE Emp_ID = eid;
  END $$
```

Stored Procedures with One Parameter: Example

Step 2: Call the stored procedure.

QUERY

```
CALL GetEmpExp(620);
```

Output:

	Emp_ID	Role_name	Dept	Experience
▶	620	Katrina	Retail	2

Stored Procedures with Two Parameter: Example

Problem Statement: You are a junior DB administrator in your organization. Your manager wants to identify employees with less than 3 years of experience and salaries less than 30000.

Objective: Create a stored procedure that takes the employee experience and salary as parameters.

Stored Procedures with Two Parameter: Example

Step 1: Create a stored procedure with the relevant employee details and keep employee experience and employee salary as the parameters.

QUERY

```
DELIMITER $$
CREATE PROCEDURE GetEmpHike(exp int, sal int)
  BEGIN
      SELECT *
      FROM Emp_Table
      WHERE Experience <= exp
      AND Salary <= sal;
  END $$
```

Stored Procedures with Two Parameter: Example

Step 2: Call the stored procedure.

QUERY

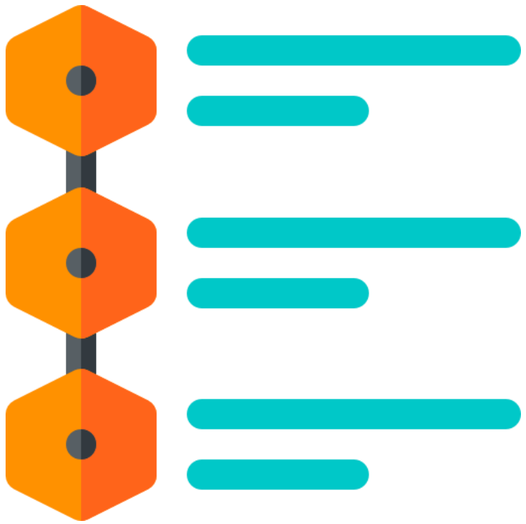
```
CALL GetEmpHike(3,30000);
```

Output:

	Emp_ID	Emp_Name	Role_name	Dept	Experience	Salary
▶	620	Katrina	Junior Data Scientist	Retail	2	21500
	366	Clair	Associate Data Scientist	Automotive	3	28000

Listing Stored Procedures

SHOW PROCEDURE STATUS statement displays all the characteristics of stored procedures.



- Returns stored procedures that have a privilege to access
- Stores stored names

Use Case: Employee Data Analysis by HR

Use Case: Employee Data Analysis by HR

Problem Statement: You work as a junior analyst at your organization. You must assist the HR department with the development of an employee information table in one of the databases so that the HR can track and retrieve their data anytime they need it.

Objective: Build the appropriate database and table for storing the HR specific data.

Download the **HR_EMP_TABLE.csv** file from the course resources section.

Use Case: Employee Data Analysis by HR

The HR department has provided a detailed description of the required table given below.

Column Name	Value Type
EMP_ID	A unique ID assigned to each employee while joining the organization
MANAGER_ID	EMP_ID of the reporting manager for the project
FIRST_NAME	First name of the employee
LAST_NAME	Last name of the employee
SALARY	Monthly salary of the employee in dollars
GENDER	Gender of the employee abbreviated as M (male), F (female), and O (others)
EXP	Overall work experience of the employee
ROLE	Employee job designation
CONTINENT	Location of the branch
COUNTRY	Country of the branch
DEPT	Department of the employee

Use Case: Employee Data Analysis by HR

Step 1: Create a database named **HR_DB** with the **CREATE DATABASE** statement.

SQL Query

```
CREATE DATABASE IF NOT EXISTS
HR_DB;
```

Use Case: Employee Data Analysis by HR

Step 2: Set **HR_DB** as the default database in MySQL with the **USE** statement.

SQL Query

```
USE HR_DB;
```

Use Case: Employee Data Analysis by HR

Step 3: Set **INNODB** as the default storage engine for HR_DB database in MySQL with the **SET** statement.

SQL Query

```
SET default_storage_engine =
INNODB;
```

Use Case: Employee Data Analysis by HR

Step 4: Create the required **EMP_RECORDS** table in the **HR_DB** database with the **CREATE TABLE** statement as given below.

SQL Query

```
CREATE TABLE IF NOT EXISTS HR_DB.EMP_RECORDS (
    EMP_ID VARCHAR(4) NOT NULL PRIMARY KEY CHECK(SUBSTR(EMP_ID,1,1) = 'E’),
    FIRST_NAME VARCHAR(100) NOT NULL,
    LAST_NAME VARCHAR(100) NOT NULL,
    GENDER VARCHAR(1) NOT NULL CHECK(GENDER IN ('M', 'F', 'O')),
    ROLE VARCHAR(100) NOT NULL,
    DEPT VARCHAR(100) NOT NULL,
    EXP INTEGER NOT NULL CHECK (EXP >= 0),
    COUNTRY VARCHAR(80) NOT NULL,
    CONTINENT VARCHAR(50) NOT NULL,
    SALARY DECIMAL(7,2) NOT NULL DEFAULT '2000.00’ CHECK(SALARY >= 2000.00),
    EMP_RATING INTEGER NOT NULL DEFAULT 1 CHECK (EMP_RATING IN (1,2,3,4,5)),
    MANAGER_ID VARCHAR(100) NOT NULL CHECK(SUBSTR(MANAGER_ID,1,1) = 'E’),
)  ENGINE=INNODB;
```

Use Case: Employee Data Analysis by HR

Step 5: Analyze the structure of the **EMP_RECORDS** table with the **DESCRIBE** statement.

SQL Query

```
DESCRIBE HR_DB.EMP_RECORDS;
```

Output:

Field	Type	Null	Key	Default	Extra
EMP_ID	varchar(4)	NO	PRI	NULL	
FIRST_NAME	varchar(100)	NO		NULL	
LAST_NAME	varchar(100)	NO		NULL	
GENDER	varchar(1)	NO		NULL	
ROLE	varchar(100)	NO		NULL	
DEPT	varchar(100)	NO		NULL	
EXP	int	NO		NULL	
COUNTRY	varchar(80)	NO		NULL	
CONTINENT	varchar(50)	NO		NULL	
SALARY	decimal(7,2)	NO		2000.00	
EMP_RATING	int	NO		1	
MANAGER_ID	varchar(100)	NO		NULL	

Use Case: Employee Data Analysis by HR

Step 6: Insert the required data from the downloaded **HR_EMP_TABLE.csv** file into the **EMP_RECORDS** table as shown below.

SQL Query

```
INSERT INTO
HR_DB.EMP_RECORDS(EMP_ID,FIRST_NAME,LAST_NAME,GENDER,ROLE,DEPT,EXP,COUNTRY,CONTIN
ENT,SALARY,EMP_RATING,MANAGER_ID)
VALUES
("E083","Patrick","Voltz","M","MANAGER","HEALTHCARE",15,"USA","NORTH
AMERICA","9500",5,"E002"),
...
...
("E403","Steve","Hoffman","M","ASSOCIATE DATA SCIENTIST","FINANCE",4,"USA","NORTH
AMERICA","5000",3,"E103");
```

Use Case: Employee Data Analysis by HR

Step 7: Analyze the data entered into the **EMP_RECORDS** table with the **SELECT** statement.

SQL Query

```
SELECT * FROM HR_DB.EMP_RECORDS;
```

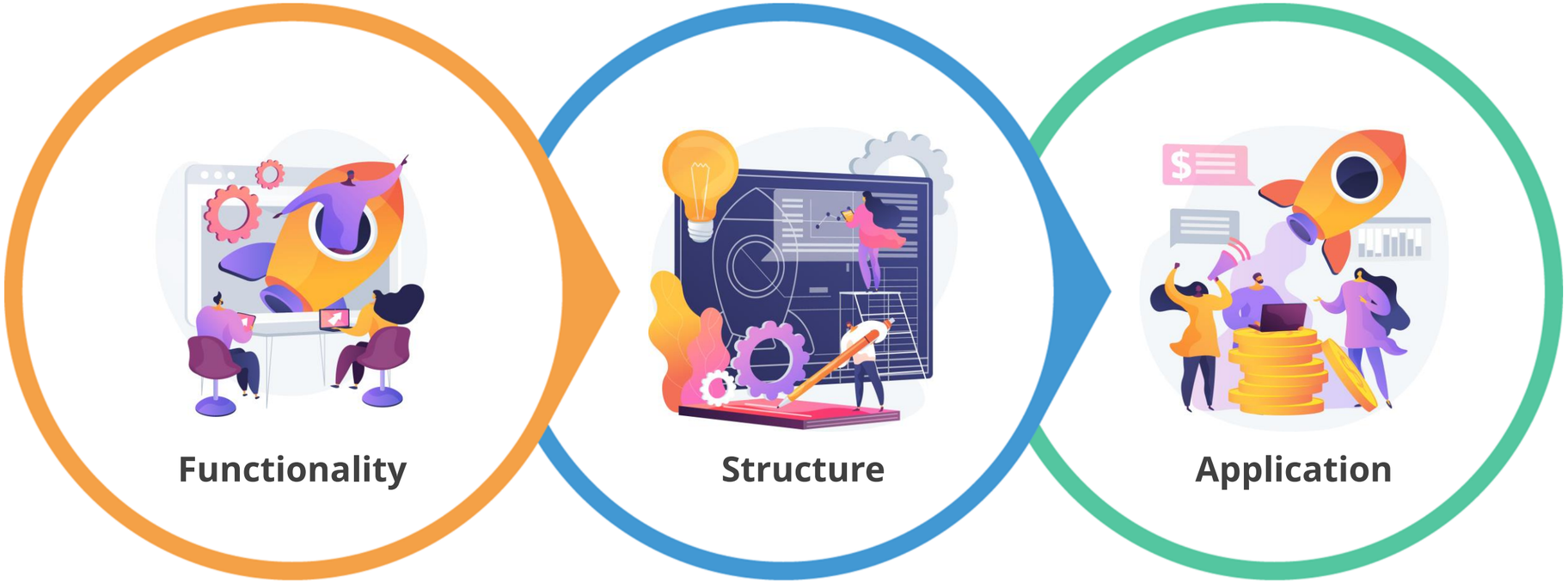
Output:

EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	EXP	COUNTRY	CONTINENT	SALARY	EMP_RATING	MANAGER_ID
E001	Arthur	Black	M	CEO	ALL	20	USA	NORTH AMERICA	16500.00	5	E001
E002	Cynthia	Brooks	F	PRESIDENT	ALL	17	CANADA	NORTH AMERICA	14500.00	5	E001
E005	Eric	Hoffman	M	LEAD DATA SCIENTIST	FINANCE	11	USA	NORTH AMERICA	8500.00	3	E103
E052	Dianna	Wilson	F	SENIOR DATA SCIENTIST	HEALTHCARE	6	CANADA	NORTH AMERICA	5500.00	5	E083
E057	Dorothy	Wilson	F	SENIOR DATA SCIENTIST	HEALTHCARE	9	USA	NORTH AMERICA	7700.00	1	E083
E083	Patrick	Voltz	M	MANAGER	HEALTHCARE	15	USA	NORTH AMERICA	9500.00	5	E002
E103	Emily	Grove	F	MANAGER	FINANCE	14	CANADA	NORTH AMERICA	10500.00	4	E002
E245	Nian	Zhen	M	SENIOR DATA SCIENTIST	RETAIL	6	CHINA	ASIA	6500.00	2	E583
E260	Roy	Collins	M	SENIOR DATA SCIENTIST	RETAIL	7	INDIA	ASIA	7000.00	3	E583
E403	Steve	Hoffman	M	ASSOCIATE DATA SCIENTIST	FINANCE	4	USA	NORTH AMERICA	5000.00	3	E103
E428	Pete	Allen	M	MANAGER	AUTOMOTIVE	14	GERMANY	EUROPE	11000.00	4	E002
E505	Chad	Wilson	M	ASSOCIATE DATA SCIENTIST	HEALTHCARE	5	CANADA	NORTH AMERICA	5000.00	2	E083
E532	Claire	Brennan	F	ASSOCIATE DATA SCIENTIST	AUTOMOTIVE	3	GERMANY	EUROPE	4300.00	1	E428
E583	Janet	Hale	F	MANAGER	RETAIL	14	COLOMBIA	SOUTH AMERICA	10000.00	2	E002
E612	Tracy	Norris	F	MANAGER	RETAIL	13	INDIA	ASIA	8500.00	4	E002
E620	Katrina	Allen	F	JUNIOR DATA SCIENTIST	RETAIL	2	INDIA	ASIA	3000.00	1	E583
E640	Jenifer	Jhones	F	JUNIOR DATA SCIENTIST	RETAIL	1	COLOMBIA	SOUTH AMERICA	2800.00	4	E583

Compound Statement

Compound Statement

Compound statement is a block that contains declarations of variables, condition handlers, cursors, loops, and conditional tests.



Compound Statement

The compound statements are enclosed within a BEGIN...END block.

Syntax

```
[begin_label:] BEGIN

    [statement_list]

END [end_label]
```

Compound Statement: Example

Problem Statement: The HR department wants to extract the manager’s details along with the number of employees reporting to the manager by using the manager’s employee ID.

Objective: Create a stored procedure that takes the manager's employee ID as input and returns the manager's basic information as well as the number of employees reporting to the manager.

Compound Statement: Example

Step 1: Create a stored procedure as shown below:

SQL Query – Par 1

```
DROP PROCEDURE IF EXISTS GetEmpCount;
DELIMITER $$
CREATE PROCEDURE GetEmpCount(IN mid VARCHAR(4))
BEGIN
    DECLARE empCount INT;
    DECLARE managerName VARCHAR(255);
    SET empCount = 0;
    SET managerName = (SELECT CONCAT(FIRST_NAME, ' ',
LAST_NAME) FROM EMP_RECORDS WHERE EMP_ID = mid);
WHILE empCount < 1 DO
        SELECT m.EMP_ID, m.FIRST_NAME, m.LAST_NAME,
        m.ROLE, m.DEPT, COUNT(e.EMP_ID) AS `EMP_COUNT`
```

SQL Query – Par 2

```
FROM EMP_RECORDS m
        LEFT JOIN EMP_RECORDS e ON m.EMP_ID =
e.MANAGER_ID
        WHERE m.ROLE IN ("MANAGER", "PRESIDENT","CEO")
        AND m.EMP_ID = mid
        GROUP BY m.EMP_ID
        ORDER BY m.EMP_ID;
        SET empCount = empCount + 1;
    END WHILE;
END $$
DELIMITER ;
```

Compound Statement: Example

Step 2: Execute this stored procedure with the EMP_ID of the manager and check the output.

SQL Query

```
CALL GetEmpCount("E083");
```

Output:

	EMP_ID	FIRST_NAME	LAST_NAME	ROLE	DEPT	EMP_COUNT
▶	E083	Patrick	Voltz	MANAGER	HEALTHCARE	3

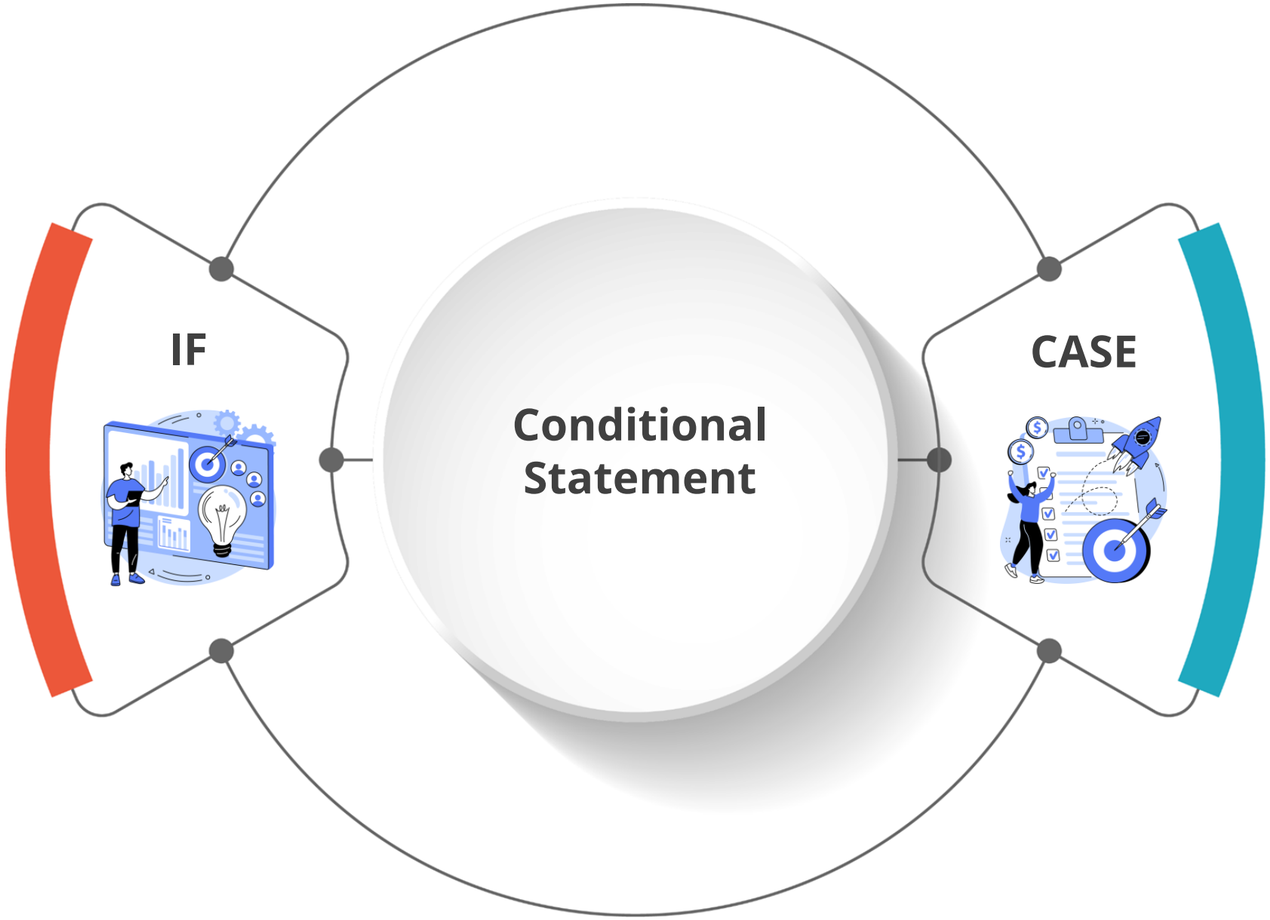
Conditional Statements

Conditional Statements

Conditional statements are used to regulate the flow of an SQL query's execution by describing the logic that will be executed if a condition is satisfied.

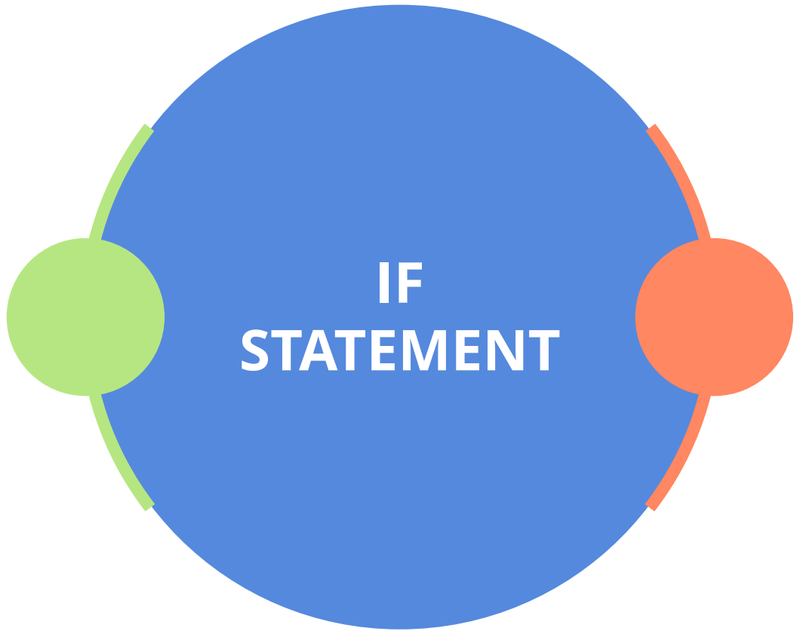
Types of Conditional Statements

There are two types of conditional statements supported by MySQL.



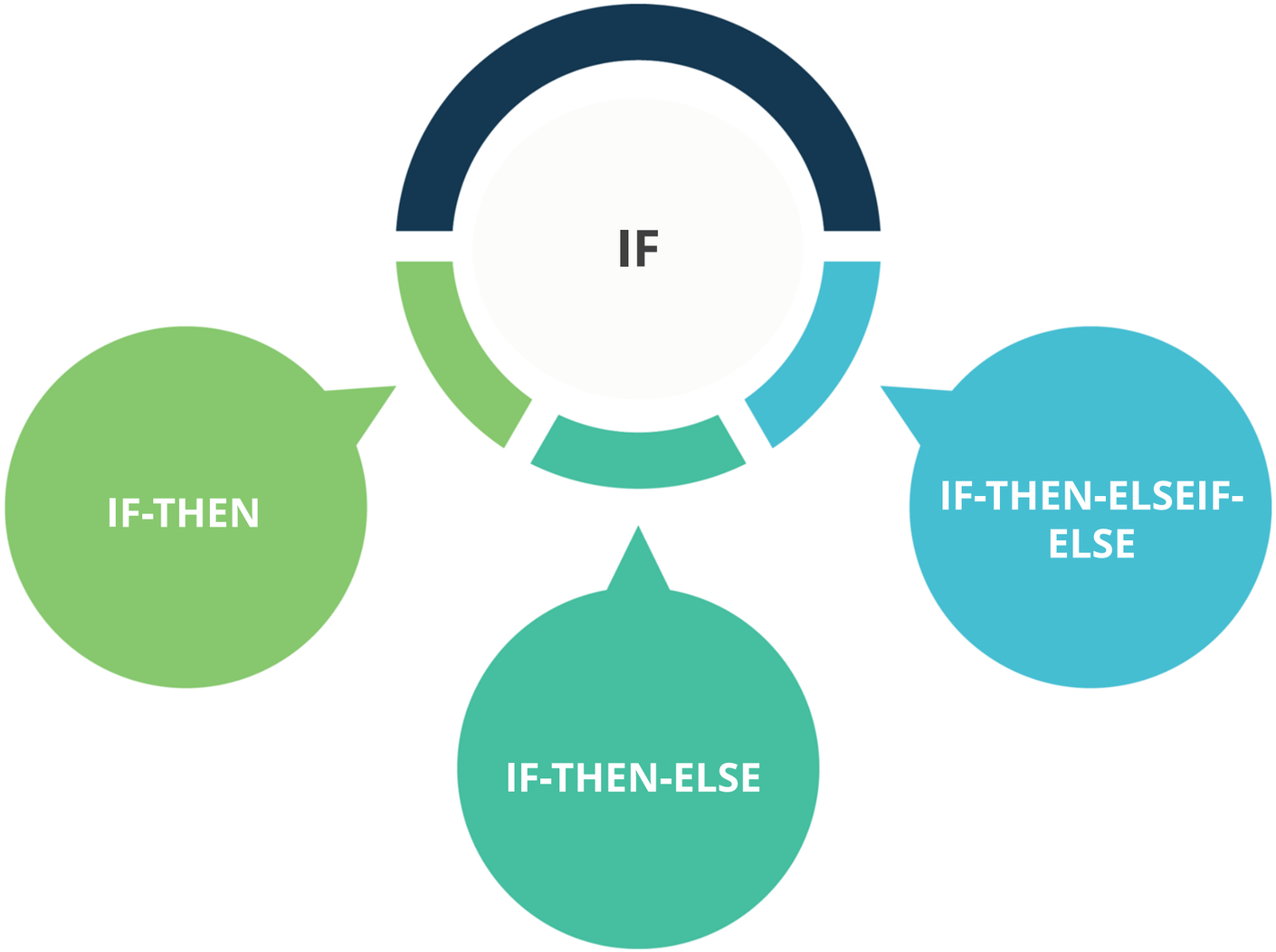
IF Statement

It is a type of control-flow statement that determines whether to execute a block of SQL code based on a specified condition.



In the IF condition, a block of SQL code is specified between the IF and END IF keywords.

Types of IF Statements



IF-THEN Statement

The IF-THEN statement executes a set of SQL statements based on a specified condition.

Syntax

```
IF condition THEN

   statement_list;

END IF;
```

IF-THEN Statement: Example

Problem Statement: The HR department wants to identify the employees who have a rating below three and are not performing well.

Objective: Create a stored procedure to determine if an employee's performance is bad depending on the rating, where a rating below three indicates bad performance.

IF-THEN Statement: Example

Step 1: Create a stored procedure as shown below:

SQL Query – Par 1

```
DROP PROCEDURE IF EXISTS getEmpScore1;
DELIMITER $$
CREATE PROCEDURE getEmpScore1(
    IN eid VARCHAR(4),
    OUT performance VARCHAR(50))
BEGIN
    DECLARE score INT DEFAULT 1;
    SELECT EMP_RATING INTO score
    FROM EMP_RECORDS WHERE EMP_ID = eid;
```

SQL Query – Par 2

```
    IF score < 3 THEN
      SET performance = "BAD";
    END IF;
END $$
```

IF-THEN Statement: Example

Step 2: Use the **EMP_ID** and a temporary global variable **@performance** to run this stored procedure, and then use this variable to examine the result.

SQL Query

```
CALL getEmpScore1("E505",
@performance);
```

SQL Query

```
SELECT @performance;
```

Output:

	@performance
▶	BAD

IF-THEN-ELSE Statement

The IF-THEN-ELSE statement executes another set of SQL statements when the condition in the IF branch does not evaluate to TRUE.

Syntax

```
IF condition THEN

   statement_list;

ELSE

   statement_list;

END IF;
```

IF-THEN-ELSE Statement

Problem Statement: The HR department needs to know the employees who have a rating below three and are not doing well as well as the ones with a rating of three or more and are performing well.

Objective: Create a stored procedure to determine if an employee's performance is good or bad depending on the rating, where a rating below three indicates bad performance and three or above indicates good performance.

IF-THEN-ELSE Statement

Step 1: Create a stored procedure as shown below.

SQL Query – Par 1

```
DROP PROCEDURE IF EXISTS getEmpScore2;
DELIMITER $$
CREATE PROCEDURE getEmpScore2(
    IN eid VARCHAR(4), OUT performance VARCHAR(50))
BEGIN
    DECLARE score INT DEFAULT 1;
    SELECT EMP_RATING INTO score
    FROM EMP_RECORDS
    WHERE EMP_ID = eid;
```

SQL Query – Par 2

```
    IF score < 3 THEN
        SET performance = "BAD";
    ELSE
        SET performance = "GOOD";
    END IF;
END$$
```

IF-THEN-ELSE Statement

Step 2: Use the **EMP_ID** and a temporary global variable **@performance** to run this stored procedure, and then use this variable to examine the result.

SQL Query

```
CALL getEmpScore2("E005",
@performance);
```

SQL Query

```
SELECT @performance;
```

Output:

	@performance
▶	GOOD

IF-THEN-ELSEIF-ELSE Statement

The IF-THEN-ELSEIF-ELSE statement conditionally executes a set of SQL statements based on multiple conditions.

Syntax

```
IF condition THEN

   statement_list;

ELSEIF elseif_condition THEN

   statement_list;

...

ELSE

   statement_list;

END IF;
```

It can have multiple ELSEIF branches.

IF-THEN-ELSEIF-ELSE Statement: Example

Problem Statement: The HR department needs to know how each employee is performing by categorizing them based on their ratings, which vary from one to five. They want to grade each employee's performance as Overachiever, Excellent Performance, Meeting Expectations, Below Expectations, and Not Achieving Any Goals.

Objective: Create a stored procedure to determine the performance of an employee based on the employee rating with an additional criteria for identifying an invalid rating using the IF-THEN-ELSEIF-ELSE statement.

IF-THEN-ELSEIF-ELSE Statement: Example

Step 1: Create a stored procedure as shown below.

SQL Query – Par 1

```
DROP PROCEDURE IF EXISTS getEmpScore3;
DELIMITER $$
CREATE PROCEDURE getEmpScore3(
    IN eid VARCHAR(4),
    OUT performance VARCHAR(50))
BEGIN
    DECLARE score INT DEFAULT 1;
    SELECT EMP_RATING INTO score FROM EMP_RECORDS
WHERE EMP_ID = eid;
    IF score = 5 THEN
        SET performance = "Over Achiever";
```

SQL Query – Par 2

```
    ELSEIF score = 4 THEN
        SET performance = "Excellent Performance";
    ELSEIF score = 3 THEN
        SET performance = "Meeting Expectation";
    ELSEIF score = 2 THEN
        SET performance = "Below Expectation";
      ELSEIF score = 1 THEN
        SET performance = "Not Achieving Any
Goals";
      ELSE
        SET performance = "Invalid Rating";
END IF;
END$$
```

IF-THEN-ELSEIF-ELSE Statement: Example

Step 2: Use the **EMP_ID** and a temporary global variable **@performance** to run this stored procedure, and then use this variable to examine the result.

SQL Query

```
CALL getEmpScore3("E005",
@performance);
```

SQL Query

```
SELECT @performance;
```

Output:

	@performance
▶	Meeting Expectation

CASE Statement

CASE Statement

It is a type of control-flow statement used in stored procedures to create conditional statements that make the code more readable and efficient.

It extends the functionality of the IF statement.

Types of CASE Statements

Simple CASE Statement

It checks for equality; however, it cannot be used to check for equality with NULL because NULL returns FALSE.

Syntax

```
CASE case_value
    WHEN when_value1 THEN statements
    WHEN when_value2 THEN statements
    ELSE
        [ELSE else_statements]

        Or,

        BEGIN
        END;
END CASE;
```

It only allows a value to be compared to a set of distinct values.

Simple CASE Statement

MySQL raises an error in absence of the ELSE clause if no conditions are satisfied.

Error

```
Case not found for CASE statement
```

The ELSE clause utilizes an empty BEGIN...END block to prevent any errors.

Simple CASE Statement: Example

Problem Statement: The HR department needs to know how each employee is performing by categorizing them based on their ratings, which vary from one to five. They want to grade each employee's performance as Overachiever, Excellent Performance, Meeting Expectations, Below Expectations, and Not Achieving Any Goals.

Objective: Create a stored procedure to determine the performance of an employee based on the employee rating with an additional criteria for identifying an invalid rating using the Simple CASE statement.

Simple CASE Statement: Example

Step 1: Create a stored procedure as shown below.

SQL Query – Par 1

```
DROP PROCEDURE IF EXISTS getEmpScore4;
DELIMITER $$
CREATE PROCEDURE getEmpScore4(
    IN eid VARCHAR(4),
    OUT performance VARCHAR(50))
BEGIN
    DECLARE score INT DEFAULT 1;
    SELECT EMP_RATING INTO score
    FROM EMP_RECORDS WHERE EMP_ID = eid;
```

SQL Query – Par 2

```
CASE score
    WHEN 5 THEN
        SET performance = "Over Achiever";
    WHEN 4 THEN
        SET performance = "Excellent Performance";
    WHEN 3 THEN
        SET performance = "Meeting Expectation";
    WHEN 2 THEN
        SET performance = "Below Expectation";
```

Simple CASE Statement: Example

Step 1: Create a stored procedure as shown below.

SQL Query – Par 3

```
    WHEN 1 THEN
        SET performance = "Not Achieving Any
Goals";
    ELSE
      BEGIN
        SET performance = "Invalid Rating";
      END;
    END CASE;
END$$
```

Simple CASE Statement: Example

Step 2: Use the **EMP_ID** and a temporary global variable **@performance** to run this stored procedure, and then use this variable to examine the result.

SQL Query

```
CALL getEmpScore4("E005",
@performance);
```

SQL Query

```
SELECT @performance;
```

Output:

	@performance
▶	Below Expectation

Searched CASE Statement

It is similar to the IF statement; however, it is considerably more readable.

Syntax

```
CASE
    WHEN search_condition1 THEN statements
    WHEN search_condition1 THEN statements
    ...
    [ELSE else_statements]

    Or,

    BEGIN
    END;
END CASE;
```

It is used for performing more complex matching such as ranges.

Searched CASE Statement

MySQL raises an error in absence of the ELSE clause if no condition evaluates to TRUE.

Error

```
Case not found for CASE statement
```

The ELSE clause utilizes an empty BEGIN...END block to prevent errors.

Searched CASE Statement: Example

Problem Statement: The HR department needs to know how each employee is performing by categorizing them based on their ratings, which vary from one to five. They want to grade each employee's performance as Overachiever, Excellent Performance, Meeting Expectations, Below Expectations, and Not Achieving Any Goals.

Objective: Create a stored procedure to determine the performance of an employee based on the employee rating with an additional criteria for identifying an invalid rating using the Searched CASE statement.

Searched CASE Statement: Example

Step 1: Create a stored procedure as shown below.

SQL Query – Par 1

```
DROP PROCEDURE IF EXISTS getEmpScore5;
DELIMITER $$
CREATE PROCEDURE getEmpScore6(
    IN eid VARCHAR(4),
    OUT performance VARCHAR(50))
BEGIN
    DECLARE score INT DEFAULT 1;
    SELECT EMP_RATING INTO score
    FROM EMP_RECORDS WHERE EMP_ID = eid;
```

SQL Query – Par 2

```
CASE
    WHEN score = 5 THEN
        SET performance = "Over Achiever";
    WHEN score = 4 THEN
        SET performance = "Excellent Performance";
    WHEN score = 3 THEN
        SET performance = "Meeting Expectation";
    WHEN score = 2 THEN
        SET performance = "Below Expectation";
```

Searched CASE Statement: Example

Step 1: Create a stored procedure as shown below.

SQL Query – Par 3

```
    WHEN score = 1 THEN
         SET performance = "Not Achieving Any
Goals";
    ELSE
      BEGIN
         SET performance = "Invalid Rating";
      END;
    END CASE;
END$$
```

Searched CASE Statement: Example

Step 2: Use the **EMP_ID** and a temporary global variable **@performance** to run this stored procedure, and then use this variable to examine the result.

SQL Query

```
CALL getEmpScore5("E005",
@performance);
```

SQL Query

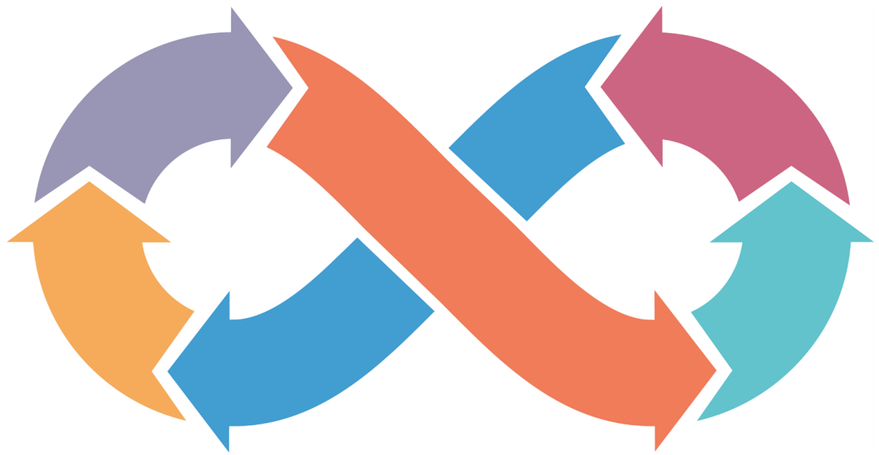
```
SELECT @performance;
```

Output:

	@performance
▶	Over Achiever

Loops in Stored Procedures

Loops in Stored Procedures



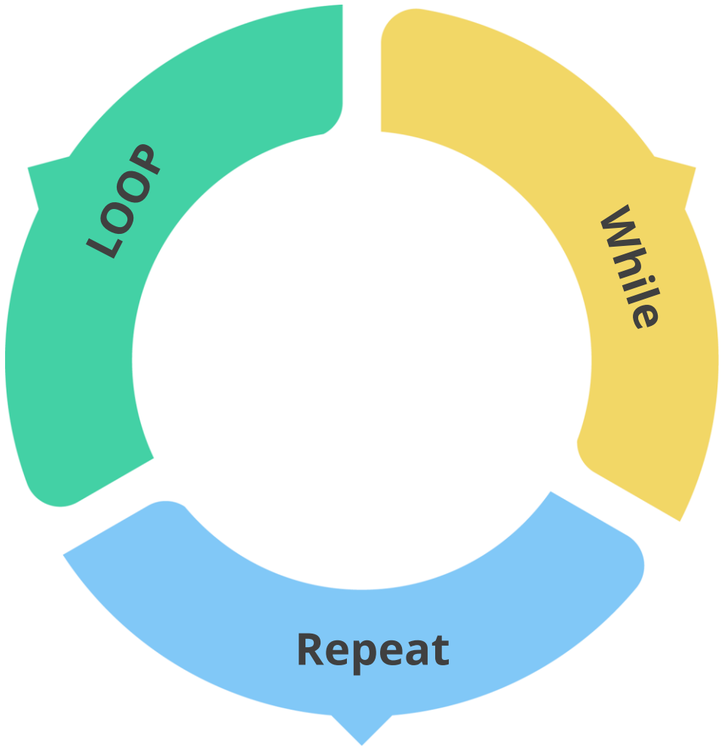
Loops, also known as iterative control statements, are a programming structure that repeats a set of instructions until a specific condition is satisfied.

They are crucial for saving time and reducing errors.

Types of Loops

There are three types of loops supported by MySQL.

Executes one or more statements repeatedly for an infinite number of times



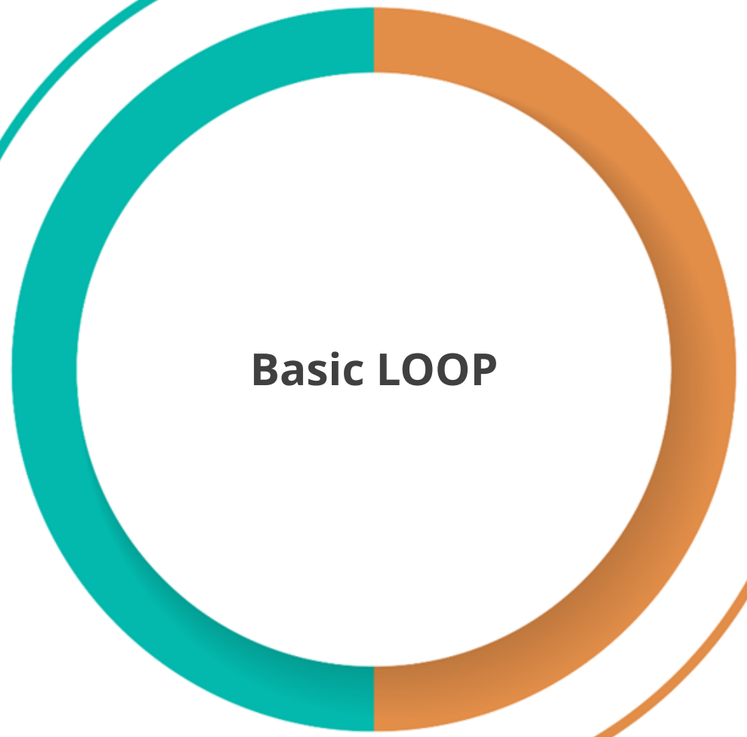
Executes one or more statements repeatedly as long as a condition is TRUE

Executes one or more statements repeatedly until a condition is satisfied

LOOP Statement

DEFINITION

Executes one or more statements repeatedly for an infinite number of times



FUNCTIONALITY

Extends the functionality of IF statement

LOOP Statement

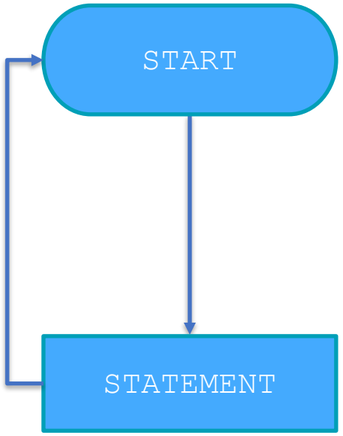
Syntax

```
[begin_label]: LOOP

    statement_list

END LOOP [end_label];
```

Flowchart



LOOP Statement: Example

Problem Statement: Your manager expects you to write a simple infinite loop in MySQL that counts even integers and adds them one after the other in a string separated by a comma.

Objective: Create a stored procedure with a simple infinite loop in MySQL that counts even integers from 1 and adds them one after the other in a string separated by a comma.

LOOP Statement: Example

Step 1: Create a stored procedure as given below.

SQL Query – Par 1

```
DROP PROCEDURE InfiniteEvenLoop;


DELIMITER $$
CREATE PROCEDURE InfiniteEvenLoop()
BEGIN
    DECLARE num  INT;
    DECLARE msg  VARCHAR(300);
    SET num = 1;
    SET msg =  '';
```

SQL Query – Par 2

```
    loop_label:  LOOP
        SET  num = num + 1;
        IF  (num mod 2) THEN
              ITERATE  loop_label;
        ELSE
              SET  msg = CONCAT(msg,num,',');
        END  IF;
    END LOOP;
    SELECT msg;
END$$
```

LOOP Statement: Example

Step 2: Execute this stored procedure with the **CALL** statement, and analyze the warning produced by MySQL as the loop is infinite.

SQL Query

```
CALL InfiniteEvenLoop();
```

Output:

#	Time	Action	Message	Duration / Fetch
1	16:25:51	CALL InfiniteEvenLoop()	Error Code: 1406. Data too long for column 'msg' at r...	0.000 sec

WHILE Loop

DEFINITION

Executes one or more statements repeatedly as long as a condition is TRUE

FUNCTIONALITY

Also known as a pretest loop because it checks the search condition before executing the statement(s)

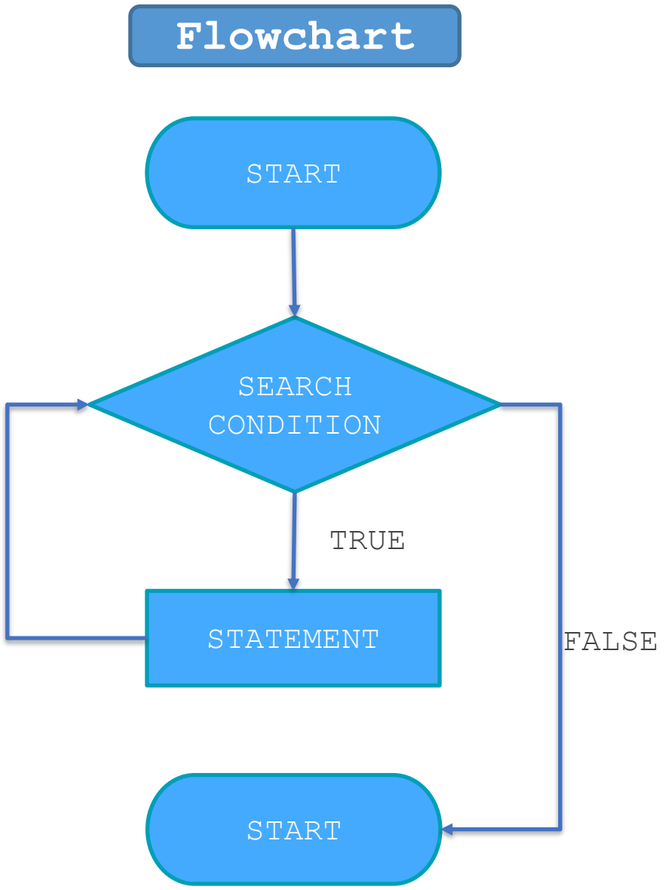
WHILE Loop

Syntax

```
[begin_label]: LOOP

    statement_list

END LOOP [end_label];
```



WHILE Loop: Example

Problem Statement: Your manager expects you to write a loop in MySQL that counts integers till 10 and adds them one after the other in a string separated by a comma.

Objective: Create a stored procedure with a while loop in MySQL that counts integers till 10 and adds them one after the other in a string separated by a comma.

WHILE Loop: Example

Step 1: Create a stored procedure as shown below:

SQL Query – Par 1

```
DROP PROCEDURE EvenWhileLoop;


DELIMITER $$
CREATE PROCEDURE EvenWhileLoop()
BEGIN
    DECLARE num  INT;
    DECLARE msg  VARCHAR(300);
    SET num = 1;
    SET msg =  '';
```

SQL Query – Par 2

```
  WHILE num <=10 DO
    SET msg = CONCAT(msg,num,',');
    SET num = num + 1;
  END WHILE;
  SELECT msg;
END $$
```

WHILE Loop: Example

Step 2: Execute this stored procedure with **CALL** statement and check the output.

SQL Query

```
CALL EvenWhileLoop();
```

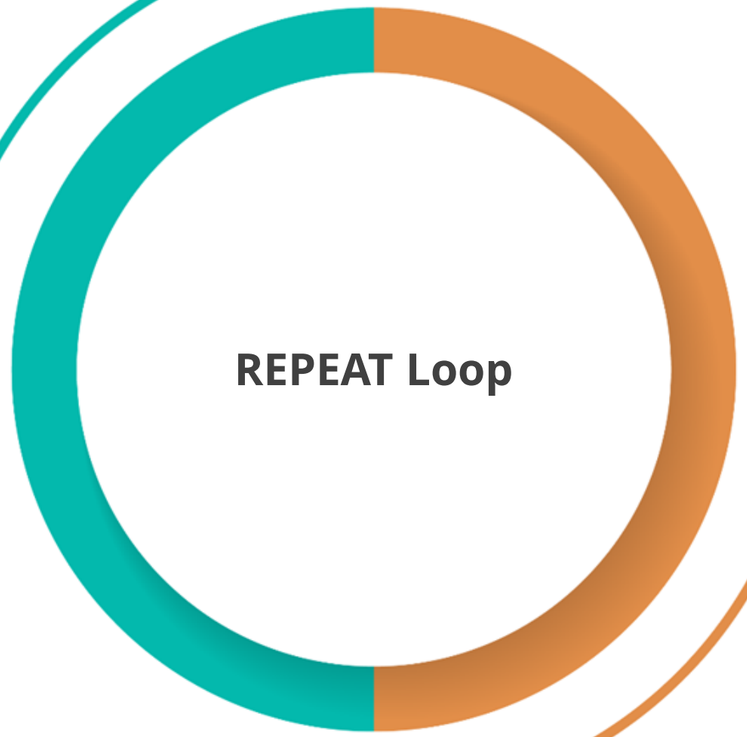
Output:

	msg
▶	1,2,3,4,5,6,7,8,9,10,

REPEAT Loop

DEFINITION

Executes one or more statements repeatedly until a condition is satisfied



FUNCTIONALITY

Also known as a post-test loop because it checks the search condition after the execution of the statement(s)

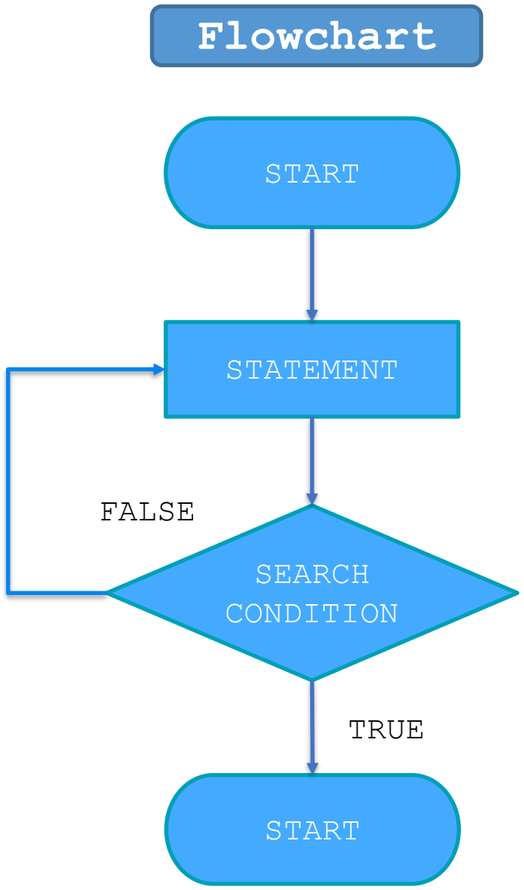
REPEAT Loop

Syntax

```
[begin_label]: LOOP

    statement_list

END LOOP [end_label];
```



REPEAT Loop: Example

Problem Statement: Your manager expects you to write a loop in MySQL that counts integers till 10 and adds them one after the other in a string separated by a comma.

Objective: Create a stored procedure with a repeat loop in MySQL that counts integers till 10 and adds them one after the other in a string separated by a comma.

REPEAT Loop: Example

Step 1: Create a stored procedure as given below.

SQL Query – Par 1

```
DROP PROCEDURE EvenRepeatLoop;

Delimiter $$
CREATE PROCEDURE EvenRepeatLoop()
BEGIN
  DECLARE num INT DEFAULT 1;
  DECLARE msg Varchar(300) DEFAULT '';
```

SQL Query – Par 2

```
  REPEAT
    SET msg = CONCAT(msg,num,',');
    SET num = num + 1;
  UNTIL num > 10
  END REPEAT;
  SELECT msg;
END $$
```

REPEAT Loop: Example

Step 2: Execute this stored procedure with **CALL** statement and check the output.

SQL Query

```
CALL EvenRepeatLoop();
```

Output:

	msg
▶	1,2,3,4,5,6,7,8,9,10,

Terminating Stored Procedures and Loops

LEAVE Statement

The LEAVE statement is used to exit a flow control which has a specific label, such as stored programs or loops.

Syntax

```
LEAVE label;
```

Using LEAVE With Stored Procedure

A LEAVE statement is used to terminate a stored procedure or function.

Syntax

```
DELIMITER //

CREATE PROCEDURE
procedure_name(parameter_list)
[label]: BEGIN
   statement_list;
   ...
   IF condition THEN
        LEAVE [label];
    END IF;
    ...
    -- other statements
END //

DELIMITER;
```

Using LEAVE With Stored Procedure: Example

Problem Statement: The HR department wants to find the employees with a rating above 3 along with their basic information.

Objective: Create a stored procedure for the basic information of an employee only if the rating is above 3.

Using LEAVE With Stored Procedure: Example

Step 1: Create a stored procedure as given below.

SQL Query – Par 1

```
DROP PROCEDURE GoodEmployeeRecord;

Delimiter $$
CREATE PROCEDURE GoodEmployeeRecord(eid VARCHAR(4))
sp: BEGIN
    DECLARE rating INT DEFAULT 0;
    SELECT EMP_RATING INTO rating
    FROM EMP_RECORDS WHERE EMP_ID = eid;
```

SQL Query – Par 2

```
    IF rating < 3 THEN
       LEAVE sp;
    END IF;
    SELECT EMP_ID, FIRST_NAME, LAST_NAME, DEPT,
SALARY
    FROM EMP_RECORDS WHERE EMP_ID = eid;
END$$
```

Using LEAVE With Stored Procedure: Example

Step 2: Execute this stored procedure with the **EMP_ID** of the employee and check the output.

SQL Query

```
CALL GoodEmployeeRecord("E083");
```

Output:

	EMP_ID	FIRST_NAME	LAST_NAME	DEPT	SALARY
▶	E083	Patrick	Voltz	HEALTHCARE	9500.00

Using LEAVE With LOOP Statement

The LEAVE statement is used in a LOOP statement when a set of statements needs to be executed at least once before the termination of the loop.

Syntax

```
[label]: LOOP
    statement_list
    ...
    -- terminate the loop
    IF condition THEN
        LEAVE [label];
    END IF;
    ...
    -- other statements
END LOOP [label];
```

Using LEAVE With LOOP Statement: Example

Problem Statement: Your manager expects you to write a simple infinite loop in MySQL that counts even integers to add them one after the other in a string separated by a comma and terminate when the integer exceeds 10.

Objective: Create a stored procedure with a simple infinite **LOOP** in MySQL that counts even integers till 10 and adds them one after the other in a string separated by a comma, terminating the loop when the integer count exceeds 10.

Using LEAVE With LOOP Statement: Example

Step 1: Create a stored procedure as given below.

SQL Query – Par 1

```
DROP PROCEDURE EvenLoop;

DELIMITER $$
CREATE PROCEDURE EvenLoop()
BEGIN
  DECLARE num  INT;
  DECLARE msg  VARCHAR(300);
  SET num = 1;
  SET msg =  '';
```

SQL Query – Par 2

```
loop_label: LOOP
    IF  num > 10 THEN
      LEAVE  loop_label;
    END  IF;
    SET  num = num + 1;
    IF  (num mod 2) THEN
      ITERATE  loop_label;
```

Using LEAVE With LOOP Statement: Example

Step 1: Create a stored procedure as given below.

SQL Query – Par 3

```
    ELSE
      SET  msg = CONCAT(msg,num,',');
    END  IF;
  END LOOP loop_label;
  SELECT msg;
END$$
```

Using LEAVE With LOOP Statement: Example

Step 2: Execute this stored procedure with the **CALL** statement, and analyze the warning produced by MySQL as the loop is infinite.

SQL Query

```
CALL EvenLoop();
```

Output:

	msg
▶	2,4,6,8,10,

Using LEAVE With WHILE Loop

A LEAVE statement is used to terminate a WHILE loop when a specific condition is satisfied before its loop condition becomes FALSE.

Syntax

```
[label:] WHILE search_condition DO
    statement_list
    ...
    -- terminate the loop
    IF condition THEN
        LEAVE [label];
    END IF;
    ...
    -- other statements
END WHILE [label];
```

Using LEAVE With WHILE Loop: Example

Problem Statement: Your manager expects you to write a loop in MySQL that produces the sum of first 20 even integers.

Objective: Create a stored procedure with a **WHILE Loop** in MySQL that counts first 20 even integers and adds them one by one, terminating the loop when the integer count exceeds the 20th even integer and return the sum.

Using LEAVE With WHILE Loop: Example

Step 1: Create a stored procedure as given below.

SQL Query – Par 1

```
DROP PROCEDURE LeaveWhileLoop;


Delimiter $$
CREATE PROCEDURE LeaveWhileLoop()
BEGIN
  DECLARE num INT;
  DECLARE sum INT;
  SET num = 1;
  SET sum = 0;
```

SQL Query – Par 2

```
  while_label: WHILE num < 200 DO
    IF num > 20 THEN
      LEAVE while_label;
    END IF;
    SET sum = sum + 2;
    SET num = num + 1;
  END WHILE while_label;
  SELECT sum;
END $$
```

Using LEAVE With WHILE Loop: Example

Step 2: Execute this stored procedure with **CALL** statement and check the output.

SQL Query

```
CALL LeaveWhileLoop();
```

Output:

	sum
▶	40

Using LEAVE With REPEAT Loop

A LEAVE statement is used to terminate a REPEAT loop when a specific condition is satisfied before its loop condition becomes TRUE.

Syntax

```
[label:] REPEAT
    statement_list
    ...
    -- terminate the loop
    IF condition THEN
        LEAVE [label];
    END IF;
    ...
    -- other statements
UNTIL search_condition
END REPEAT [label];
```

Using LEAVE With WHILE Loop: Example

Problem Statement: Your manager expects you to write a loop in MySQL that produces the sum of first 20 even integers.

Objective: Create a stored procedure with a **REPEAT Loop** in MySQL that counts first 20 even integers and adds them one by one, terminating the loop when the integer count exceeds the 20th even integer and return the sum.

Using LEAVE With REPEAT Loop: Example

Step 1: Create a stored procedure as given below.

SQL Query – Par 1

```
DROP PROCEDURE LeaveRepeatLoop;

Delimiter $$
CREATE PROCEDURE LeaveRepeatLoop()
BEGIN
  DECLARE num INT;
  DECLARE sum INT;
  SET num = 1;
  SET sum = 0;
```

SQL Query – Par 2

```
  repeat_label: REPEAT
    SET sum = sum + 2;
    SET num = num + 1;
    IF num > 20 THEN
      LEAVE repeat_label;
    END IF;
  UNTIL num > 200
  END REPEAT repeat_label;
  SELECT sum;
END$$
```

Using LEAVE With REPEAT Loop: Example

Step 2: Execute this stored procedure with **CALL** statement and check the output.

SQL Query

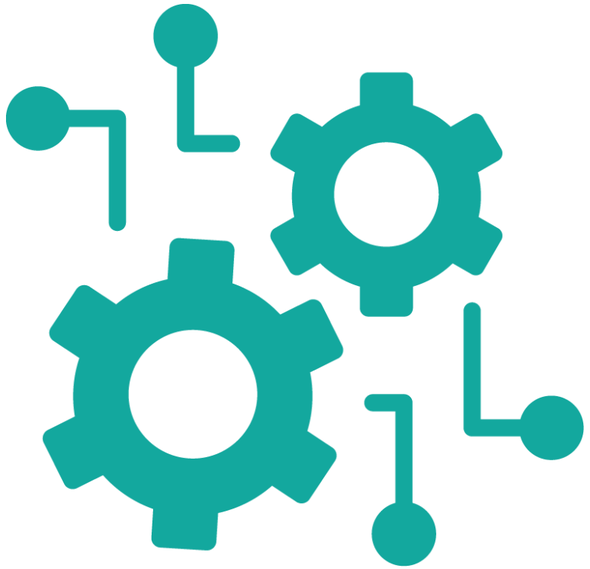
```
CALL LeaveRepeatLoop();
```

Output:

	sum
▶	40

Stored Functions in Stored Procedures

Stored Functions in Stored Procedures



A stored function is a stored program that returns a single value.

It is a subroutine that may be accessed by programs that use a relational database management system.

Stored Functions in Stored Procedures

The CREATE FUNCTION statement is used for creating a stored function and user-defined functions.

Stored Functions in Stored Procedures

Syntax for creating a stored function:

```
DELIMITER $$

CREATE FUNCTION fun_name(fun_parameter(s))
RETURNS datatype
[NOT] {Characteristics}
fun_body;
```

Stored Functions in Stored Procedures

Parameters	Description
fun_name	The name of the stored function that the user wants to create
fun_parameter	The list of parameters used by the function body
datatype	The data type that returns value of the function
fun_body	The set of SQL statements that performs the operations

Stored Functions in Stored Procedures

Example:

```
DELIMITER //

CREATE FUNCTION no_of_years(date1 date) RETURNS
int
BEGIN
 DECLARE date2 DATE;
  Select current_date()into date2;
  RETURN year(date2)-year(date1);
END //
DELIMITER ;
SELECT no_of_years(Start_Date) AS
years_difference;
```

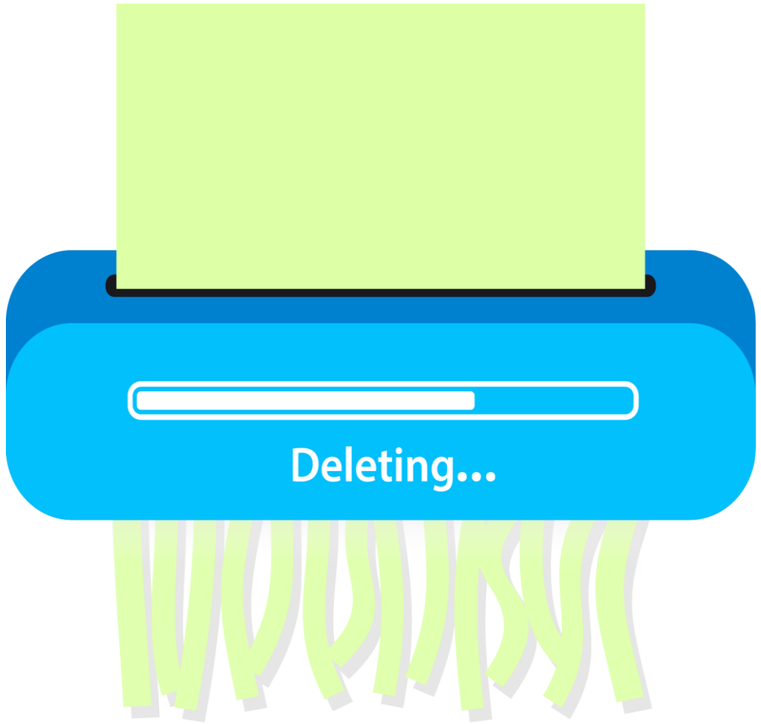
EMP_ID	Fname	Lname	Start_Date
1	Ravi	Kumar	2000-01-09
2	Slim	Shady	2005-09-11
3	Michael	Scott	2002-04-12
4	Travis	Baker	2004-06-11

Stored Functions in Stored Procedures

Output:

EMP_ID	Fname	Lname	Years
1	Ravi	Kumar	21
2	Slim	Shady	16
3	Michael	Scott	19
4	Travis	Baker	17

Stored Functions in Stored Procedures



To remove a stored function, we use the DROP FUNCTION.

Syntax for removing a stored function:

```
DROP FUNCTION [IF EXISTS] function_name;
```

Stored Functions in Stored Procedures

The list of all stored functions from databases can be accessed by using SHOW FUNCTION STATUS.

Stored Functions in Stored Procedures

Syntax for listing stored function:

```
SHOW FUNCTION STATUS
[LIKE 'pattern' | WHERE search_condition];
```

Stored Functions in Stored Procedures

Syntax for listing stored function using the data dictionary:

```
SELECT
    routine_name
FROM
    information_schema.routines
WHERE
    routine_type = 'FUNCTION'
        AND routine_schema = '<database_name>';
```

Problem Statement

Problem Scenario: You are a junior database administrator. Your manager has asked you to perform different operations using **stored function** on the **emp_details** table with the schema named as **sys**.

Objective: You are required to extract the first and last names, department, and designation based on the experience of the employees.

Solution

Query for designation

```
DELIMITER $$
drop function sys.Customer_details1;
CREATE FUNCTION Customer_details1(experience int)
RETURNS VARCHAR(2255) DETERMINISTIC
BEGIN DECLARE customer_details1 VARCHAR(2255);
IF experience <= 2 THEN SET customer_details1 = 'JUNIOR DATA SCIENTIST';
ELSEIF experience <= 5 THEN SET customer_details1 = 'ASSOCIATE DATA SCIENTIST';
ELSEIF experience <= 10 THEN SET customer_details1 = 'SENIOR DATA SCIENTIST';
ELSEIF experience <= 12 THEN SET customer_details1 = 'LEAD DATA SCIENTIST';
ELSEIF experience > 12 THEN SET customer_details1 = 'MANAGER'
END IF; RETURN (customer_details1); END$$ DELIMITER $$;
SELECT first_name, last_name, department, Customer_details1(experience) as designation FROM
sys.emp_details ORDER BY experience;
```

Output

After executing the query, the list of names, department, experience, and designation are shown as the following output:

first_name	last_name	department	designation	experience
Jenifer	Jhones	RETAIL	JUNIOR DATA SCIENTIST	1
Katrina	Allen	RETAIL	JUNIOR DATA SCIENTIST	2
David	Smith	RETAIL	ASSOCIATE DATA SCIENTIST	3
Claire	Brennan	AUTOMOTIVE	ASSOCIATE DATA SCIENTIST	3
Steve	Hoffman	FINANCE	ASSOCIATE DATA SCIENTIST	4
Chad	Wilson	HEALTHCARE	ASSOCIATE DATA SCIENTIST	5
Dianna	Wilson	HEALTHCARE	SENIOR DATA SCIENTIST	6
Nian	Zhen	RETAIL	SENIOR DATA SCIENTIST	6
Karene	Nowak	AUTOMOTIVE	SENIOR DATA SCIENTIST	8
Dorothy	Wilson	HEALTHCARE	SENIOR DATA SCIENTIST	9
Eric	Hoffman	FINANCE	LEAD DATA SCIENTIST	11
Slim	Shady	FINANCE	LEAD DATA SCIENTIST	11
William	Butler	AUTOMOTIVE	LEAD DATA SCIENTIST	12
Tracy	Norris	RETAIL	MANAGER	13
Emily	Grove	FINANCE	MANAGER	14

Problem Statement

Problem Scenario: You are a junior database administrator. Your manager has asked you to perform different operations using **stored function** on the **emp_details** table with the schema named as **sys**.

Objective: You are required to extract the names and status of the projects using stored procedure with stored functions.

Solution

Query for stored function

```
DELIMITER $$
CREATE FUNCTION Customer_details1(project_id VARCHAR(225))
RETURNS VARCHAR(2255) DETERMINISTIC
BEGIN DECLARE customer_details1 VARCHAR(2255);
IF project_id = 'P103' THEN SET customer_details1 = 'Drug Discovery';
ELSEIF project_id = 'P105' THEN SET customer_details1 = 'Fraud Detection';
ELSEIF project_id = 'P109'  THEN SET customer_details1 = 'Market Basket Analysis';
ELSEIF project_id = 'P201'  THEN SET customer_details1 = 'Self Driving Cars';
ELSEIF project_id = 'P204' THEN SET customer_details1 = 'Supply Chain Management';
ELSEIF project_id = 'P208' THEN SET customer_details1 = 'Algorithmic Trading';
ELSEIF project_id = 'P302' THEN SET customer_details1 = 'Early Detection of Lung Cancer';
ELSEIF project_id = 'P406' THEN SET customer_details1 = 'Customer Sentiment Analysis';
END IF; RETURN (customer_details1); END$$
DELIMITER $$;
```

Solution

Query for using stored procedure

```
DELIMITER $$

CREATE PROCEDURE GetCustomerDetail() BEGIN

SELECT project_id, status , Customer_details1(project_id) as
project_name FROM sys.project_details ORDER BY project_id;

END$$ DELIMITER ; call GetCustomerDetail();
```

Output

After executing the query, the list of **project_id**, **project_name**, and **status** are shown as the following output:

	project_id	project_name	status
▶	P103	Drug Discovery	DONE
	P105	Fraud Detection	DONE
	P109	Market Basket Analysis	DELAYED
	P201	Self Driving Cars	YTS
	P204	Supply Chain Management	WIP
	P208	Algorithmic Trading	YTS
	P302	Early Detection of Lung Cancer	YTS
	P406	Customer Sentiment Analysis	WIP

Difference Between Stored Functions and Stored Procedures

Stored Functions

- A function has a return type and returns a value.
- Functions cannot be used with data manipulation queries. Only select queries are allowed in functions.
- A function does not allow output parameters.
- Stored procedures cannot be called from a function.

Stored Procedures

- A procedure does not have a return type. It returns values using the OUT parameters.
- DML queries such as insert, update, and select can be used with procedures.
- A procedure allows both input and output parameters.
- Functions can be called from a stored procedure.

Assisted Practice: Stored Procedures

Duration: 20 mins

Problem statement: You are working for a company that deals with geographical data. Most of the SQL work done here is via stored procedures. You are an analyst and have been asked to create a stored procedure to pull all North American countries using the COUNTRY table.

Assisted Practice: Stored Procedures

Steps to be performed:

Step 01: Create a view containing the columns **COUNTRY_ID**, **COUNTRY_NAME**, and **CONTINENT_ID** and name it **"COUNTRY“**

SQL Query

```
DROP TABLE IF EXISTS COUNTRY;
CREATE TABLE COUNTRY (
COUNTRY_ID INT,
COUNTRY_NAME TEXT,
CONTINENT_ID INT,
CONTINENT_NAME TEXT
);
```

Assisted Practice: Stored Procedures

Output:

	#	Time	Action	Message
✔	1	10:06:39	DROP TABLE IF EXISTS customer	0 row(s) affected
✔	2	10:06:39	CREATE TABLE customer (ORDER_ID INTEGER, CUST_ID T...	0 row(s) affected

Assisted Practice: Stored Procedures

Step 02: Insert values in the **COUNTRY** table

SQL Query

```
INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (1, 'Ukraine', 3, 'Europe');
INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (2, 'France', 3, 'Europe'); INSERT
COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (3, 'Germany', 3, 'Europe'); INSERT
COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (4, 'Italy', 3, 'Europe'); INSERT COUNTRY
(COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (5, 'United States', 2, 'North America'); INSERT
COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (6, 'Bosnia and Herzegovina', 3,
'Europe'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (7, 'United Kingdom',
3, 'Europe'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (8, 'Japan', 1,
'Asia'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (9, 'Indonesia', 1,
'Asia'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (10, 'Vietnam', 1,
'Asia'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (11, 'Russia', 3,
'Europe'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (12, 'Switzerland', 3,
'Europe'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (13, 'Cuba', 2, 'North
America');
```

Assisted Practice: Stored Procedures

Output:

	#	Time	Action	Message
✔	1	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	2	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	3	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	4	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	5	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	6	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	7	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	8	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	9	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	10	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	11	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	12	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	13	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected

Assisted Practice: Stored Procedures

Step 03: Display the **COUNTRY** table

SQL Query

```
SELECT * FROM COUNTRY
```

Assisted Practice: Stored Procedures

Output:

#	COUNTRY_ID	COUNTRY_NAME	CONTINENT_ID	CONTINENT_NAM
1	1	Ukraine	3	Europe
2	2	France	3	Europe
3	3	Germany	3	Europe
4	4	Italy	3	Europe
5	5	United States	2	North America
6	6	Bosnia and Herzegovina	3	Europe
7	7	United Kingdom	3	Europe
8	8	Japan	1	Asia
9	9	Indonesia	1	Asia
10	10	Vietnam	1	Asia
11	11	Russia	3	Europe
12	12	Switzerland	3	Europe
13	13	Cuba	2	North America

Assisted Practice: Stored Procedures

Step 04: Write a query to create a stored procedure and name it **"SP_COUNTRIES_NA"**

SQL Query

```
DELIMITER &&
CREATE PROCEDURE SP_COUNTRIES_NA()
BEGIN
SELECT COUNTRY_NAME FROM COUNTRY WHERE CONTINENT_NAME="North America";
END &&
CALL SP_COUNTRIES_NA();
```

Assisted Practice: Stored Procedures

Output:

#	COUNTRY_NAME
1	United States
2	Cuba
3	United States
4	Cuba

Assisted Practice: Stored Procedures with One Parameter

Duration: 20 mins

Problem statement: You are working for a company that deals with geographical data. Most of the SQL work done here is via stored procedures. You are an analyst and have been asked to create a stored procedure to pull the count of distinct countries in a continent, passed in the procedure argument, using the **COUNTRY** table.

Assisted Practice: Stored Procedures with One Parameter

ASSISTED PRACTICE

Steps to be performed:

Step 01: Create a view containing the columns **COUNTRY_ID**, **COUNTRY_NAME**, and **CONTINENT_ID** and name it **"COUNTRY“**

SQL Query

```
DROP TABLE IF EXISTS COUNTRY;
CREATE TABLE COUNTRY (
COUNTRY_ID INT,
COUNTRY_NAME TEXT,
CONTINENT_ID INT,
CONTINENT_NAME TEXT);
```

Assisted Practice: Stored Procedures with One Parameter

Output:

	#	Time	Action	Message
✔	1	10:06:39	DROP TABLE IF EXISTS customer	0 row(s) affected
✔	2	10:06:39	CREATE TABLE customer (ORDER_ID INTEGER, CUST_ID T...	0 row(s) affected

Assisted Practice: Stored Procedures with One Parameter

Step 02: Insert values in the **COUNTRY** table

SQL Query

```
INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (1, 'Ukraine', 3, 'Europe');
INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (2, 'France', 3, 'Europe'); INSERT
COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (3, 'Germany', 3, 'Europe'); INSERT
COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (4, 'Italy', 3, 'Europe'); INSERT COUNTRY
(COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (5, 'United States', 2, 'North America'); INSERT
COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (6, 'Bosnia and Herzegovina', 3,
'Europe'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (7, 'United Kingdom',
3, 'Europe'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (8, 'Japan', 1,
'Asia'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (9, 'Indonesia', 1,
'Asia'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (10, 'Vietnam', 1,
'Asia'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (11, 'Russia', 3,
'Europe'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (12, 'Switzerland', 3,
'Europe'); INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTINENT_ID, CONTINENT_NAME) VALUES (13, 'Cuba', 2, 'North
America');
```

Assisted Practice: Stored Procedures with One Parameter

Output:

	#	Time	Action	Message
✔	1	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	2	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	3	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	4	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	5	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	6	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	7	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	8	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	9	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	10	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	11	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	12	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected
✔	13	18:24:51	INSERT COUNTRY (COUNTRY_ID, COUNTRY_NAME, CONTI...	1 row(s) affected

Assisted Practice: Stored Procedures with One Parameter

Step 03: Display the **COUNTRY** table

SQL Query

```
SELECT * FROM COUNTRY
```

Assisted Practice: Stored Procedures with One Parameter

Output:

#	COUNTRY_ID	COUNTRY_NAME	CONTINENT_ID	CONTINENT_NAM
1	1	Ukraine	3	Europe
2	2	France	3	Europe
3	3	Germany	3	Europe
4	4	Italy	3	Europe
5	5	United States	2	North America
6	6	Bosnia and Herzegovina	3	Europe
7	7	United Kingdom	3	Europe
8	8	Japan	1	Asia
9	9	Indonesia	1	Asia
10	10	Vietnam	1	Asia
11	11	Russia	3	Europe
12	12	Switzerland	3	Europe
13	13	Cuba	2	North America

Assisted Practice: Stored Procedures with One Parameter

Step 04: Write a query to create a stored procedure and name it **"SP_COUNTRIES_OF_CONTINENT"**

SQL Query

```
DELIMITER &&
CREATE PROCEDURE SP_COUNTRIES_OF_CONTINENT(IN CONT_NAME TEXT)
BEGIN
SELECT COUNT(DISTINCT COUNTRY_NAME) FROM COUNTRY WHERE CONTINENT_NAME=CONT_NAME;
END &&

CALL SP_COUNTRIES_OF_CONTINENT('Asia');

CALL SP_COUNTRIES_OF_CONTINENT('North America');
```

Assisted Practice: Stored Procedures with One Parameter

Output:

#	COUNT(DISTINCT COUNTRY_N
1	3

#	COUNT(DISTINCT COUNTRY_N
1	2

Assisted Practice: Stored Procedures with Multiple Values

Duration: 20 mins

Problem statement: You work for an ed-tech startup that curates and delivers courses for experienced professionals. You have been asked to create a procedure that classifies the courses into three categories:

- StudentCount less than or equal to 1000
- StudentCount greater than 1000 and StudentCount less than or equal to 5000
- StudentCount greater than 5000

Assisted Practice: Stored Procedures with Multiple Values

Steps to be performed:

Step 01: Create a view with the columns **CourseID**, **CourseName**, and **StudentCount** and name it **"Course"**

CREATE

```
CREATE TABLE Course (
CourseID INT,
CourseName TEXT,
StudentCount INT);
```

Assisted Practice: Stored Procedures with Multiple Values

Output:

	#	Time	Action	Message
✔	1	04:03:49	CREATE TABLE Course (CourseID INT, CourseName TEXT, ...	0 row(s) affected

Assisted Practice: Stored Procedures with Multiple Values

Step 02: Insert values in the **Course** table

Query

```
INSERT INTO Course(CourseID, CourseName, StudentCount) VALUES(1,'Data
Science',100034),
(2,'PMP',2500),
(3,'Agile',4003),
(4,'ITIL',387),
(5,'Blockchain',7876);
```

Assisted Practice: Stored Procedures with Multiple Values

Output:

	#	Time	Action	Message
✔	1	04:04:42	INSERT INTO Course(CourseID, CourseName, StudentCoun...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0

Assisted Practice: Stored Procedures with Multiple Values

Step 03: Display the **Course** table

Query

```
SELECT * FROM Course;
```

Assisted Practice: Stored Procedures with Multiple Values

Output:

#	CourseID	CourseName	StudentCoun
1	1	Data Science	100034
2	2	PMP	2500
3	3	Agile	4003
4	4	ITIL	387
5	5	Blockchain	7876

Assisted Practice: Stored Procedures with Multiple Values

Step 04: Write a query to create a stored procedure and name it **"CLASSIFY_COURSE"**

Query

```
DELIMITER &&
CREATE PROCEDURE CLASSIFY_COURSE(IN id INT, OUT CATEGORY TEXT)
BEGIN DECLARE C INT DEFAULT 1;
SELECT StudentCount INTO C FROM Course WHERE CourseID=id;
IF C<=1000 THEN
SET CATEGORY = "Low Demand";
ELSEIF C>1000 AND C<=5000 THEN
SET CATEGORY = "Mid Demand";
ELSE SET CATEGORY = "High Demand";
END IF; END &&
CALL CLASSIFY_COURSE(1, @CATEGORY);
SELECT @CATEGORY;
```

Assisted Practice: Stored Procedures with Multiple Values

Output:

#	@CATEGORY
1	High Demand

Assisted Practice: Stored Procedures with Multiple Values

Step 05: Write a query to create a stored procedure called **"CLASSIFY_COURSE"**

Test Case 01

```
DELIMITER &&
CREATE PROCEDURE CLASSIFY_COURSE(IN id INT, OUT CATEGORY TEXT)
BEGIN DECLARE C INT DEFAULT 1;
SELECT StudentCount INTO C FROM Course WHERE CourseID=id;
IF C<=1000 THEN
SET CATEGORY = "Low Demand";
ELSEIF C>1000 AND C<=5000 THEN
SET CATEGORY = "Mid Demand";
ELSE SET CATEGORY = "High Demand";
END IF; END &&
CALL CLASSIFY_COURSE(3, @CATEGORY);
SELECT @CATEGORY;
```

Assisted Practice: Stored Procedures with Multiple Values

Output:

#	@CATEGORY
1	Mid Demand

Knowledge Check

Knowledge Check 1

Which of the following defines a stored procedure in SQL?

A. Block of functions

B. Group of SQL statements

C. Collection of views

D. None of the above

Knowledge Check 1

Which of the following defines a stored procedure in SQL?

A. Block of functions

B. Group of SQL statements

C. Collection of views

D. None of the above

The correct answer is **B**

A stored procedure is a logical unit in a database that groups one or more precompiled SQL statements contained within the BEGIN and END keywords in the stored procedure's body.

Knowledge Check 2

What is the purpose of the SQL CASE statement?

A. To filter rows in a SELECT query

B. To perform calculations on numeric columns

C. To conditionally change the value of a column

D. To join tables in a query

Knowledge Check 2

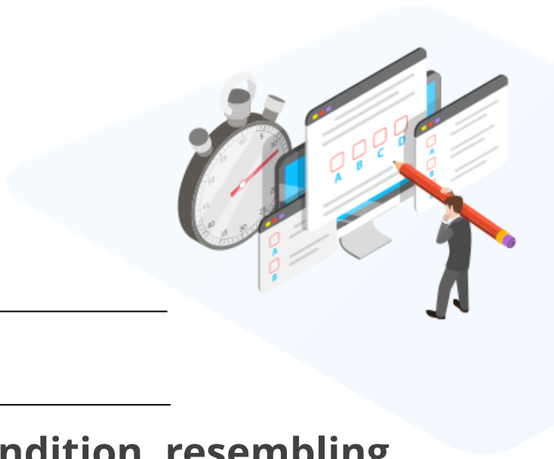
What is the purpose of the SQL CASE statement?

A. To filter rows in a SELECT query

B. To perform calculations on numeric columns

C. To conditionally change the value of a column

D. To join tables in a query



The correct answer is **C**

The CASE expression evaluates conditions sequentially and yields a value upon the first met condition, resembling an if-then-else statement. Therefore, once a condition becomes true, the evaluation halts, and the corresponding result is returned.

Knowledge Check 3

Which of the following is an escape character in SQL?

A. Period

B. Comma

C. Colon

D. Backslash

Knowledge Check 3

Which of the following is an escape character in SQL?

A. Period

B. Comma

C. Colon

D. Backslash

The correct answer is **D**

Backslash is the escape character in MySQL.

Knowledge Check 4

What is the keyword used to create a stored procedure?

A. DECLARE PROCEDURE

B. SET PROCEDURE

C. CREATE PROCEDURE

D. ASSIGN PROCEDURE

Knowledge Check 4

What is the keyword used to create a stored procedure?

A. DECLARE PROCEDURE

B. SET PROCEDURE

C. CREATE PROCEDURE

D. ASSIGN PROCEDURE

The correct answer is **C**

CREATE PROCEDURE is the keyword used to create stored procedure in MySQL.

Knowledge Check 5	Which of the following repeats a set of instructions until a specific condition is reached?

A. Loops

B. Control structures

C. Repeat statement

D. Stored procedures

Knowledge Check 5	**Which of the following repeats a set of instructions until a specific condition is reached?**

A. Loops

B. Control structures

C. Repeat statement

D. Stored procedures

The correct answer is **C**

Repeat statements are used to repeat a set of instructions until a specific condition is met.

Knowledge Check 6	Which characteristics are used to control the privileges of execution of a stored object? Select all that apply.

A. DEFINER

B. SET

C. SQL SECURITY

D. DECLARE

Knowledge Check 6	**Which characteristics are used to control the privileges of execution of a stored object? Select all that apply.**

A. DEFINER

B. SET

C. SQL SECURITY

D. DECLARE

The correct answer are **A, C**

DEFINER and SQL SECURITY characteristics are used to control the privileges of execution of a stored object.

Lesson-End Project: Payroll Calculation

Problem statement:

You are a part of the HR department in a company, and you have been asked to calculate the monthly payout for each employee based on their experience and performance.

Objective:

The objective is to design a database to retrieve the detailed salary divisions of each employee in the organization.

Note: Download the **employee_datasets.csv** and **department_datasets.csv** files from **Course Resources** to perform the required tasks

LESSON-END PROJECT

Lesson-End Project: Payroll Calculation

Tasks to be performed:

1. Write a query to create the **employee** and **department** tables
2. Write a query to insert values into the **employee** and **department** tables
3. Write a query to create a view of the **employee** and **department** tables
4. Write a query to display the first and last names of every employee in the **employee** table whose salary is greater than the average salary of the employees listed in the **SQL basics** table
5. Write a query to change the delimiter to //

LESSON-END PROJECT

Lesson-End Project: Payroll Calculation

Tasks to be performed:

6. Write a query to create a stored procedure in the **employee** table for every employee whose salary is greater than or equal to 250,000
7. Write a query to execute the stored procedure
8. Write a query to create and execute a stored procedure with one parameter using the **order by** function in descending order of the salary earned

Note: Download the solution document from the **Course Resources** section and follow the steps given in the document

LESSON-END PROJECT

Key Takeaways

- A stored procedure is executed using the CALL keyword.
- The CREATE PROCEDURE keyword is used to create stored procedures, which can contain one or more comma-separated parameters.
- IF and CASE are the two types of conditional statements that govern the execution of a SQL query.
- A trigger is a set of actions that are executed automatically when a specific change operation is performed on a specific table.