STORED FUNCTIONS CREATION

```
CREATE FUNCTION function_name
returns data_type
return function_statement
```

- Example of a stored function that has one SQL statement

```
CREATE FUNCTION world_record_count ()
RETURNS INT
RETURN SELECT COUNT(*) FROM Country;
```

- The stored function is invoked by:

```
SELECT function_name();
```

STORED PROCEDURE CREATION

```
CREATE PROCEDURE procedure_name
procedure_statement
```

- Example of a stored procedure that has one SQL statement

```
CREATE PROCEDURE world_record_count ()
SELECT 'country count ', COUNT(*) FROM
Country;
```

- The stored procedure is invoked by:

```
CALL procedure_name();
```

COMPOUND STATEMENT

- Example of a stored procedure that has more than one SQL statement

```
DELIMITER //

CREATE PROCEDURE world_record_count ()

BEGIN

SELECT 'country count ', COUNT(*) FROM
country;

SELECT 'city count ', COUNT(*) FROM city;
SELECT 'CountryLanguage count', COUNT(*) FROM
CountryLanguage;

END//

DELIMITER ;
```

VARIABLES IN STORED PROCEDURES

- Variables are used in stored procedure to store the immediate result.

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

- The **variable name** should follow the naming convention and should not be the same name of table or column in a database
- The **data type** of the variable, it can be any primitive type which MySQL supports such as `INT`, `VARCHAR` and `DATETIME`…along with the data type is the size of the variable. When you declare a variable, its initial value is `NULL`.

ASSIGNING VARIABLES

- Once you declared a variable, you can start using it. To assign other value to a variable you can use
  - `SET` statement
  - `SELECT … INTO` to assign a query result to a variable.

```
DECLARE total_count INT DEFAULT 0;
SET total_count = 10;

DECLARE total_products INT DEFAULT 0;
SELECT COUNT(*) INTO total_products FROM
products;
```

CONTROL FLOW

- The two common flow controls are:
  - **Choices** – statements that are obeyed under certain conditions. In MySQL, these are represented in the **IF** and **CASE** statements.
  - **Loops** - statements that are obeyed repeatedly. In MySQL these are represented in the **REPEAT**, **WHILE** and **LOOP** statements


```
IF (test_condition)
THEN ...
ELSEIF (test_condition)
THEN ...
ELSE ...
END IF
```

CASE

```
CASE case_value
WHEN when_value
THEN ...
ELSE ...
END CASE
```

```
CASE WHEN test_condition
THEN ...
ELSE ...
END CASE
```

REPEAT

```
mylabel: REPEAT

...

UNTIL test_condition

END REPEAT mylabel
```

REPEAT

```
DELIMITER $$
DROP PROCEDURE IF EXISTS RepeatLoopProc$$
CREATE PROCEDURE RepeatLoopProc()
BEGIN
DECLARE x INT;
DECLARE str VARCHAR(255);
SET x = 1;
SET str = '';
REPEAT
SET str = CONCAT(str,x,',');
SET x = x + 1;
UNTIL x > 5
END REPEAT;
SELECT str;
END$$
DELIMITER ;
```

WHILE

```
mylabel: WHILE test_condition DO

...

END WHILE mylabel
```

WHILE

```
DELIMITER $$
DROP PROCEDURE IF EXISTS WhileLoopProc$$
CREATE PROCEDURE WhileLoopProc()
BEGIN
DECLARE x INT;
DECLARE str VARCHAR(255);
SET x = 1;
SET str = '';
WHILE x <= 5 DO
SET str = CONCAT(str,x,',');
SET x = x + 1;
END WHILE;
SELECT str;
END
$$ DELIMITER ;
```

LOOP

```
mylabel: LOOP

...

[LEAVE | ITERATE ] mylabel;

END LOOP mylabel
```

LOOP

```
DELIMITER $$
CREATE PROCEDURE LOOPLoopProc()
BEGIN
DECLARE x INT;
DECLARE str VARCHAR(255);
SET x = 1;
SET str = '';
loop_label: LOOP
IF x > 10
THEN  LEAVE loop_label;
END IF;
SET x = x + 1;
IF (x mod 2)
THEN  ITERATE loop_label;
ELSE
SET str = CONCAT(str,x,',');
END IF;
END LOOP;
SELECT str;
END$$
DELIMITER ;
```