Stored Procedures:

When you use MySQL Workbench or the mysql shell to execute a query on the MySQL Server, the server processes the query and returns the result set.

If you intend to save this query on the database server for later execution, one way to achieve this is by using a stored procedure.

Sample:

```
DELIMITER $$


CREATE PROCEDURE GetCustomers()
BEGIN
	SELECT
		customerName,
		city,
		state,
		postalCode,
		country
	FROM
		customers
	ORDER BY customerName;
END$$
DELIMITER ;
```

By definition, a stored procedure is a set of declarative SQL statements stored within the MySQL Server. In this example, we have just created a stored procedure named GetCustomers().

After saving the stored procedure, you can invoke it by using the CALL statement:

```
CALL GetCustomers();
```

Summary

- A stored procedure is a wrapper of a set of SQL statements stored in the MySQL database server.

- The advantages of stored procedures include reduced network traffic, enhanced code reusability, improved security through controlled access, streamlined implementation of business logic, and the ability to grant specific privileges to applications without exposing underlying database structures.
- The disadvantages of stored procedures include increased memory usage for each connection, challenges in debugging due to a lack of dedicated tools, and the necessity for a specialized skill set, which not all application developers may possess, leading to potential difficulties in both development and maintenance processes.

DELIMITER Uses:

- When you want to execute multiple SQL statements, you use the semicolon (;) to separate two statements, as shown in the following example:
- SELECT * FROM products;
- SELECT * FROM customers;
- Code language: SQL (Structured Query Language) (sql)
- A MySQL client program, such as MySQL Workbench or the mysql program, uses the default delimiter (;) to separate statements and execute each separately.
- However, a stored procedure consists of multiple statements separated by a semicolon (;).
- If you use a MySQL client program to define a stored procedure that contains semicolons, the MySQL client program will not treat the entire stored procedure as a single statement; instead, it will recognize it as multiple statements.
- Therefore, it is necessary to temporarily redefine the delimiter so that you can pass the entire stored procedure to the server as a single statement.
- To redefine the default delimiter, you use the DELIMITER command as follows:
- DELIMITER delimiter_character

```
DELIMITER $$


CREATE PROCEDURE CreatePersonTable()

BEGIN

  -- drop persons table

  DROP TABLE IF EXISTS persons;


  -- create persons table
```

```
    CREATE TABLE persons(
        id INT AUTO_INCREMENT PRIMARY KEY,
        first_name VARCHAR(255) NOT NULL,
        last_name VARCHAR(255) NOT NULL
    );

    -- insert data into the persons table
    INSERT INTO persons(first_name, last_name)
    VALUES('John','Doe'),
                    ('Jane','Doe');

    -- retrieve data from the persons table
    SELECT id, first_name, last_name
    FROM persons;
END $$

DELIMITER ;
```

In above code:

- First, change the default delimiter to $$.
- Second, use the semicolon (;) in the body of the stored procedure and $$ after the END keyword to end the stored procedure.
- Third, revert to the default delimiter(;).

IN and OUT Parameters:

```
DELIMITER $$

CREATE PROCEDURE GetOrderCountByStatus (
        IN  orderStatus VARCHAR(25),
        OUT total INT
```

```
)
BEGIN
	SELECT COUNT(orderNumber)
	INTO total
	FROM orders
	WHERE status = orderStatus;
END$$

DELIMITER ;
------------------------------------------------------------
CALL GetOrderCountByStatus('Shipped',@total);
SELECT @total;
------------------------------------------------------------
+--------+
| @total |
+--------+
|    303 |
+--------+
1 row in set (0.00 sec)
```

INOUT Parameter:

```
DELIMITER $$

CREATE PROCEDURE SetCounter(
	INOUT counter INT,
   IN inc INT
)
BEGIN
	SET counter = counter + inc;
```

```
END$$


DELIMITER ;

-----------------------------------

SET @counter = 1;

CALL SetCounter(@counter,1); -- 2

CALL SetCounter(@counter,1); -- 3

CALL SetCounter(@counter,5); -- 8

SELECT @counter; -- 8

-----------------------------------------
```