**What are stored procedures?**

Stored procedures are ways to store one or more lines of queries as one single query. They have many benefits. It is good for saving time typing multiple queries over and over. Apparently, each query is more data traffic being sent over a network. When using a stored procedure, it is as if only one query is being sent. Stored procedures can also function as a means of security, since they can pull data from tables without allowing access to the remaining data in those tables. These stored procedures can also be made with parameters, allowing you to change their function depending on the parameters passed through when calling on the stored procedure. To set them up properly, you must first change the delimiter (keystrokes that signify the termination of a line of code for a query) from ; to something else. Then when writing the queries in the stored procedure, the normal delimiter of ; will not end the input of the stored procedure.

**What are some logical constructs that can be used in a stored procedure (****i.e. loops, conditionals, parameters, etc…)?**

There are three different types of parameters used in stored procedures: IN, OUT, and INOUT. IN parameters are created inside the parenthesis by stating IN, a chosen name for the type of column being used (the variable here), and the column’s data type. Then it is called upon by passing in a variable of that column’s data type. I tried my hand at creating an IN parameter using the employees schema and the employees table. I wanted to see a list of all the columns from employees where the hire date was after ‘1993-01-01'. I had to do some trial and error, hence the DROP PROCEDURE. Here is the query I used to create the stored procedure and call upon it.  
DROP PROCEDURE sp\_hire\_date\_after;

DELIMITER //

CREATE PROCEDURE sp\_hire\_date\_after (IN date DATE)

BEGIN

SELECT \* FROM employees WHERE hire\_date > date;

END //

DELIMITER ;

CALL sp\_hire\_date\_after ('1993-01-01');

OUT parameters will put out data based on the type called in the parameter. In the parameters it starts with OUT, followed by the name chosen for where the output data will be referenced, and then the data type of that output data. (For clarification, the data type of the output data does not necessarily mean the data type of, for example, the data being counted.) In the select statement, the name of the output data referenced is placed after INTO. For the queries I created, I wanted to count the total number of female employees in the employees table. Even though the data type for the column gender is CHAR, the data type in the parameter is an INT because it is counting the total number of rows where the CHAR is ‘F’ in this example. Here is my example I created.

DROP PROCEDURE sp\_gender\_total;

DELIMITER //

CREATE PROCEDURE sp\_gender\_total (OUT totalGender INT)

BEGIN

SELECT COUNT(gender) INTO totalGender FROM employees WHERE gender = 'F';

END //

DELIMITER ;

CALL sp\_gender\_total(@FemaleEmployees);

SELECT @FemaleEmployees AS "Female Employees";  
  
INOUT parameters will take an argument passed into the parameter (like an IN) and then output data (like an OUT) based on the argument passed into the IN variable. To slightly tweak my last example, my new stored procedure will no longer have the searched for gender be female. Instead, it will allow for flexibility. In this case I will be passing in ‘M’ as the argument. It is structured like so: INOUT, followed by the output name and data type, followed by the variable name and data type and length. Here is my example.

DROP PROCEDURE sp\_variableTotalGender;

DELIMITER //

CREATE PROCEDURE sp\_variableTotalGender (INOUT totalGender INT, genderType CHAR(1))

BEGIN

SELECT COUNT(gender) INTO totalGender FROM employees WHERE gender = genderType;

END //

DELIMITER ;

CALL sp\_variableTotalGender(@TotalEmployees,'M');

SELECT @TotalEmployees AS "Total Employees of Gender Type: Male";

**References**

1. <https://www.sqlshack.com/learn-mysql-the-basics-of-mysql-stored-procedures/>