STORED PROCEDURE

1. A Procedure (often called as Stored Procedure) is a collection of pre-complied SQL statements stored inside the database.
2. A Procedure always contains a name, parameter lists and SQL statements.
3. Stored Procedures are fast in terms of performance and are reusable.
4. A procedure is always secure. The database administrator can grant permissions to applications that access stored procedures in the database without giving any permissions on the database tables.

How to Create a Stored Procedure

- First of all there are three variants of Stored Procedures.

a. A simple Procedure : Every time you call the procedure it will do the same job.

b. Procedure with IN Parameter : User has the control to give inputs while executing the procedure.

c. Procedure with OUT Parameter: Whatever results your procedure is generating you can store that in a variable for further usage.

•Syntax for a Simple Procedure

```
DELIMITER //
CREATE PROCEDURE PROCEDURE_NAME()
BEGIN
WRITE YOUR SQL STATEMENTS I.E. WHAT YOU WANT AS A RESULT WHEN YOU EXECUTE THIS PROCEDURE;
END //
DELIMITER ;
```

Simple Procedure

Let's suppose I have an employees table and I want to create a procedure that every time I call that procedure it should just display the complete data.

```
Delimiter //
Create Procedure display_data()
Begin
Select * from employees;
End //
Delimiter ;
```

Note: The usual delimiter in SQL is a semi-colon it means that we end every query in SQL with semi-colon. Now, as we know that a procedure ends with a END statement and in the procedure we write some SQL statements.

So, we change the delimiter in the beginning to any symbol you like and we mention the same symbol in front of END as well for the system to understand that the query has to stop where you find the same symbol that we declared in front of DELIMITER in the beginning.

As SQL statement in the procedure will also end with a semi colon and if you mention the same delimiter again in front of END as well system will get confused as to where the query should stop.

Procedure with IN Parameter

Now, let's suppose I have an employees table and I want to create a procedure where I can give the condition on the salary and the job_id and accordingly the procedure displays the result.

It means the procedure will always look for two inputs at the time of execution. This is called as Procedure with IN (In means Input from the user) Parameter.

You need to mention two things in the IN parameter – 1. The variable name (any name of your choice)

2. The data type of the variable (system will accordingly accept the input).

```
Delimiter //
Create Procedure display_data(IN var1 varchar(20), IN var2 int unsigned)
Begin
Select * from employees where job_id = var1 and salary > var2;
End //
Delimiter ;
```

While calling the procedure we need to write in below manner

```
CALL display_data('IT_PROG',9000);
```

It will display all those rows where the person is a IT_PROG and the salary is greater than 9000.

User can change the input as per his/her requirement.

MySQL User- Defined Variables

Now, before we understand about Procedure with OUT parameter we need to understand what are variables and how do we create the same in MySQL.

Anything which is capable of holding information and is subjected to change is a variable

.

Too Technical don't worry - In simple terms a column name in excel is a variable in Data world.

Let's suppose you have some 100 queries where you have given some conditions on salary. Now, you want to change the salary value in all those 100 queries in one go. One option if you have lot of time with you go to all those queries one by one and change the salary value in the condition. Alternatively, what you can do is create a user defined variable and use that user defined variable in all those 100 queries.
To create a user defined variable you need to make use of @ in MySQL. See the syntax below.

```
SET @SAL = 10000;
```

Change the value as per your requirement and execute it again.

I have created a user defined variable by the name SAL and gave it a value of 10000. Now, wherever I will be giving condition on salary instead of the value I will use this variable. See the below syntax as a sample.

```
SELECT * FROM EMPLOYEES WHERE SALARY > @SAL;
```

All the places where this variable is being used will take the updated value.

Procedure with OUT parameter

Let's suppose you want to write a procedure which while calling should find the average salary of the employees and you want to store that result in some variable as you would like to use the same in some other queries for further analysis. This can be achieved by writing a Procedure with OUT parameter.

```
Delimiter //
Create Procedure display_data(OUT var1 decimal(10,3))
Begin
Select avg(salary) into var1 from employees;
End //
Delimiter ;
```

When you call the above procedure you need to call in the below manner

```
CALL display_data(@var1);
```

As the average value that the procedure generated is now stored in the variable var1 and you can now use the same variable in n number of queries instead of writing the value manually every time.

Procedure with INOUT parameter

Let's suppose you want to write a procedure where you would like to take an input from the user plus at the same time you would like to store the result generated in a variable. This is called as Procedure with INOUT parameter.

```
Delimiter //
Create Procedure display_data(IN var1 varchar(20), OUT var2 decimal(10,3))
Begin
Select avg(salary) into var2 from employees
Group by  job_id
having job_id = var1;
End //
Delimiter ;
```

When you call the above procedure you need to call in the below manner

```
CALL display_data('it_prog',@var2);
```

Then, to see what is stored in var2 write the query as - (select @var2);

You will find out that var2 contains the average salary of the people who are designated as 'it_prog' in your dataset.