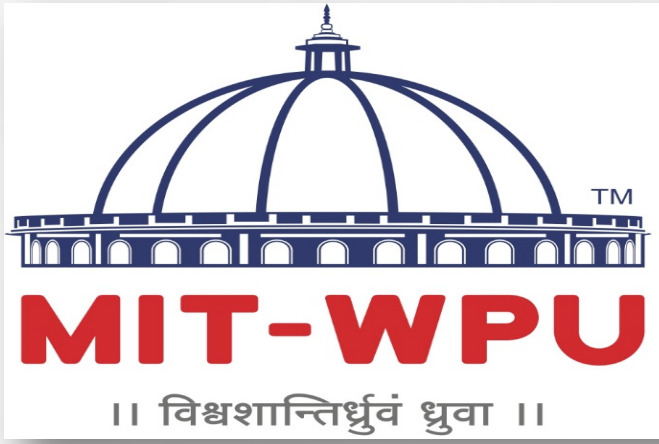


Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

DBMS- Lab 6

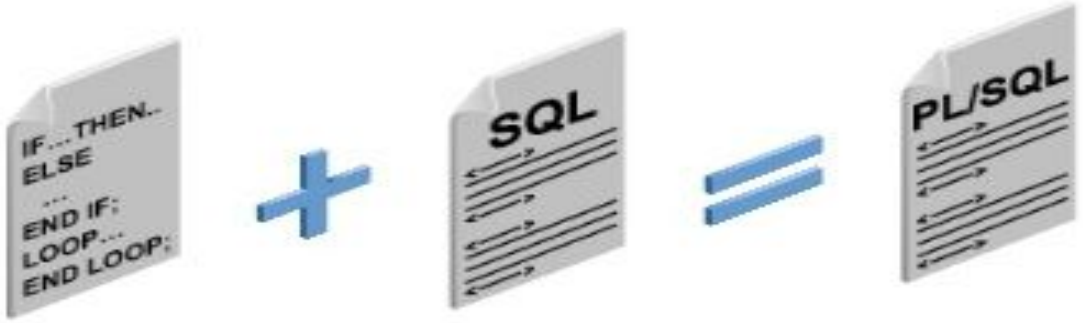
PL SQL Functions and Procedures

School of Computer Engineering and technology

What is PL/SQL

PL/SQL:

- Stands for Procedural Language extension to SQL
- Is Oracle Corporation's standard data access language for relational databases
- Seamlessly integrates procedural constructs with SQL



Stored Function

PL/SQL Functions

- Functions are declared using the following syntax:

```
Create function <function-name> (param_1, …, param_k)
        returns <return_type>
        [not] deterministic          allow optimization if same output
                    for the same input (use RAND not deterministic )
Begin
    -- execution code
end;
```

where param is:

```
        <param_name> <param_type>
```

Deterministic and Non- deterministic Functions

- A deterministic function always returns the same result for the same input parameters whereas a non-deterministic function returns different results for the same input parameters.
- If you don't use DETERMINISTIC or NOT DETERMINISTIC, MySQL uses the NOT DETERMINISTIC option by default.
- **rand()** is nondeterministic function. That means we do not know what it will return ahead of time.

```
DELIMITER $$
CREATE FUNCTION myrand() RETURNS INT
    DETERMINISTIC
BEGIN
    RETURN round(rand()*10000, 0);
END$$
```

PL/SQL Functions – Example 1

- Define a function that, given the name of a department, returns the count of the number of instructors in that department.

```
create function dept_count (dept_name varchar(20))
        returns integer
    begin
        declare d_count  integer;
            select count (* ) into d_count
            from instructor
            where instructor.dept_name = dept_name
        return d_count;
    end
```

Example 1 (Cont)..

- The function dept_count can be used to find the department names and budget of all departments with more than 12 instructors.

```
select dept_name, budget
from department
where dept_count (dept_name ) > 12
```

Example 2

- A function that returns the level of a customer based on credit limit. We use the IF statement to determine the credit limit.

```
DELIMITER $$

CREATE FUNCTION CustomerLevel(p_creditLimit double) RETURNS VARCHAR(10)
    DETERMINISTIC
BEGIN
    DECLARE lvl varchar(10);

    IF p_creditLimit > 50000 THEN
 SET lvl = 'PLATINUM';
    ELSEIF (p_creditLimit <= 50000 AND p_creditLimit >= 10000) THEN
        SET lvl = 'GOLD';
    ELSEIF p_creditLimit < 10000 THEN
        SET lvl = 'SILVER';
    END IF;

 RETURN (lvl);
END
```

Example 2 (Cont..)

- **Calling function:**
- we can call the CustomerLevel() in a SELECT statement as follows:

```
1  SELECT
2      customerName,
3      CustomerLevel(creditLimit)
4  FROM
5      customers
6  ORDER BY
7      customerName;
```

Output:

	customerName	CustomerLevel(creditLimit)
▶	Alpha Cognac	PLATINUM
	American Souvenirs Inc	SILVER
	Amica Models & Co.	PLATINUM
	ANG Resellers	SILVER
	Anna's Decorations, Ltd	PLATINUM
	Anton Designs, Ltd.	SILVER

Example 3

```
mysql> select * from employee;
+----+-------+----------+--------+---------------+------+
| id | name  | superid  | salary | bdate         | dno  |
+----+-------+----------+--------+---------------+------+
|  1 | john  |        3 | 100000 | 1960-01-01    |    1 |
|  2 | mary  |        3 |  50000 | 1964-12-01    |    3 |
|  3 | bob   |     NULL |  80000 | 1974-02-07    |    3 |
|  4 | tom   |        1 |  50000 | 1970-01-17    |    2 |
|  5 | bill  |     NULL |   NULL | 1985-01-20    |    1 |
+----+-------+----------+--------+---------------+------+
5 rows in set (0.00 sec)

mysql> delimiter |
mysql> create function giveRaise (oldval double, amount double
    -> returns double
    -> deterministic
    -> begin
    ->         declare newval double;
    ->         set newval = oldval * (1 + amount);
    ->         return newval;
    -> end |
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```

Example 3 (cont..)

```
mysql> select name, salary, giveRaise(salary, 0.1) as newsal
    -> from employee;
+------+--------+--------+
| name | salary | newsal |
+------+--------+--------+
| john | 100000 | 110000 |
| mary |  50000 |  55000 |
| bob  |  80000 |  88000 |
| tom  |  50000 |  55000 |
| bill |   NULL |   NULL |
+------+--------+--------+
5 rows in set (0.00 sec)
```

Example 4

```
DELIMITER //
CREATE FUNCTION no_of_years(date1 date) RETURNS int
DETERMINISTIC
BEGIN
 DECLARE date2 DATE;
  Select current_date()into date2;
  RETURN year(date2)-year(date1);
END //
select no_of_years('2010-04-03');
```

Stored Procedures

Stored Procedures in MySQL

- A stored procedure contains a sequence of SQL commands stored in the database catalog so that it can be invoked later by a program
- Stored procedures are declared using the following syntax:

```
Create Procedure <proc-name>
        (param_spec1, param_spec2, …, param_specn )
begin
    -- execution code
end;
```

where each param_spec is of the form:

[in | out | inout] <param_name> <param_type>

- in mode: allows you to pass values into the procedure,
- out mode: allows you to pass value back from procedure to the calling program

Example 1 – No parameters

- The GetAllProducts() stored procedure selects all products from the products table.

```
mysql> use classicmodels;
Database changed
mysql> DELIMITER //
mysql> CREATE PROCEDURE GetAllProducts()
    -> BEGIN
    -> SELECT * FROM products;
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql>
```

Example 1 (Cont..)

- **Calling Procedure:**

```
CALL GetAllProducts();
```

- **Output:**

productCode	productName	productLine	productScale
S10_1678	1969 Harley Davidson Ultimate Chopper	Motorcycles	1:10
S10_1949	1952 Alpine Renault 1300	Classic Cars	1:10
S10_2016	1996 Moto Guzzi 1100i	Motorcycles	1:10
S10_4698	2003 Harley-Davidson Eagle Drag Bike	Motorcycles	1:10
S10_4757	1972 Alfa Romeo GTA	Classic Cars	1:10

Example 2 (with IN parameter)

```
mysql> select * from employee;
+----+------+---------+--------+------------+------+
| id | name | superid | salary | bdate      | dno  |
+----+------+---------+--------+------------+------+
|  1 | john |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob  |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom  |       1 |  50000 | 1978-01-17 |    2 |
|  5 | bill |    NULL |   NULL | 1985-01-20 |    1 |
+----+------+---------+--------+------------+------+
```

```
mysql> select * from department;
+---------+-------------+
| dnumber | dname       |
+---------+-------------+
|       1 | Payroll     |
|       2 | TechSupport |
|       3 | Research    |
+---------+-------------+
```

- Suppose we want to keep track of the total salaries of employees working for each

```
mysql> create table deptsal as
    -> select dnumber, 0 as totalsalary from department;
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |           0 |
|       2 |           0 |
|       3 |           0 |
+---------+-------------+
```

We need to write a procedure to update the salaries in the deptsal table

Example 2 (Cont..)

```
mysql> delimiter //
mysql> create procedure updateSalary (IN param1 int)
    -> begin
    ->     update deptsal
    ->     set totalsalary = (select sum(salary) from employee where dno = param1)
    ->     where dnumber = param1;
    -> end; //
Query OK, 0 rows affected (0.01 sec)
```

1. Define a procedure called updateSalary which takes as input a department number.
2. The body of the procedure is an SQL command to update the totalsalary column of the deptsal table.

Example 2 (Cont..)

Step 3: Call the procedure to update the totalsalary for each department

```
mysql> call updateSalary(1);
Query OK, 0 rows affected (0.00 sec)

mysql> call updateSalary(2);
Query OK, 1 row affected (0.00 sec)

mysql> call updateSalary(3);
Query OK, 1 row affected (0.00 sec)
```

Example 2 (Cont..)

Step 4: Show the updated total salary in the deptsal table

```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

Example 3 (with OUT Parameter)

- The following example shows a simple stored procedure that uses an OUT parameter.
- Within the procedure MySQL MAX() function retrieves maximum salary from MAX_SALARY of jobs table.

```
mysql> CREATE PROCEDURE my_proc_OUT (OUT highest_salary INT)
-> BEGIN
-> SELECT MAX(MAX_SALARY) INTO highest_salary FROM JOBS;
-> END$$
Query OK, 0 rows affected (0.00 sec)
```

(Cont..)

- **Procedure Call:**
- mysql> CALL my_proc_OUT(@M)$$
- Query OK, 1 row affected (0.03 sec)
- mysql< SELECT @M$$

- **Output:**

```
+-------+
| @M    |
+-------+
| 40000 |
+-------+
1 row in set (0.00 sec)
```

Example 4 (with INOUT Parameter)

- The following example shows a simple stored procedure that uses an INOUT parameter.
- 'count' is the INOUT parameter, which can store and return values and 'increment' is the IN parameter, which accepts the values from user.

```
mysql> DELIMITER // ;
mysql> Create PROCEDURE counter(INOUT count INT, IN increment INT)
    -> BEGIN
    -> SET count = count + increment;
    -> END //
Query OK, 0 rows affected (0.03 sec)
```

Example 4 (Cont..)

Function Call:

```
mysql> DELIMITER ;
mysql> SET @counter = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL counter(@Counter, 1);
Query OK, 0 rows affected (0.00 sec)

mysql> Select @Counter;
+----------+
| @Counter |
+----------+
| 1        |
+----------+
1 row in set (0.00 sec)
```

Stored Procedures (Cont..)

- Use show procedure status to display the list of stored procedures you have

```
mysql> show procedure status;
+------+---------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
| Db   | Name          | Type      | Definer | Modified            | Created             | Security_
type | Comment | character_set_client | collation_connection | Database Collation |
+------+---------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
| ptan | updateSalary0 | PROCEDURE | ptan@%  | 2010-03-16 12:21:55 | 2010-03-16 12:21:55 | DEFINER
     |         | latin1               | latin1_swedish_ci    | latin1_swedish_ci  |
+------+---------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
1 row in set (0.02 sec)
```

- Use drop procedure to remove a stored procedure

```
mysql> drop procedure updateSalary;
Query OK, 0 rows affected (0.00 sec)
```

Language Constructs for Procedures & Functions

- SQL supports constructs that gives it almost all the power of a general-purpose programming language.
  - Warning: most database systems implement their own variant of the standard syntax below.
- Compound statement: **begin … end**,
  - May contain multiple SQL statements between **begin** and **end**.
  - Local variables can be declared within a compound statements

Language Constructs

- **CASE Statement**

```
CASE  case_expression
  WHEN when_expression_1 THEN commands
  WHEN when_expression_2 THEN commands
  ...
  ELSE commands
END CASE;
```

- **While** and **repeat** statements:

while *boolean expression* **do**
sequence of statements ;
end while

repeat
sequence of statements ;
until *boolean expression*
end repeat

Language Constructs (Cont.)

- **Loop, Leave and Iterate statements…**
  - Permits iteration over all results of a query.

```
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;
```

Language Constructs (Cont.)

- **Conditional statements (if-then-else)**

```
IF expression THEN
  statements;
ELSE
  else-statements;
END IF;
```

Example:

```
IF creditlim > 50000 THEN
SET p_customerLevel = 'PLATINUM';
  ELSEIF (creditlim <= 50000 AND creditlim >= 10000) THEN
    SET p_customerLevel = 'GOLD';
  ELSEIF creditlim < 10000 THEN
    SET p_customerLevel = 'SILVER';
  END IF;
```

Batch 1 Exercise 1

Consider the employee table

Employee (emp_id, first_name,last_name,hiredate)

Write a stored procedure to take the emp_id as input parameter. Procedure must raise the salary of an employee based on following conditions

If experience is less than 2 years then salary raise is 5%

If experience is between 2 to 5 years then raise is 7%

If experience is more than 5 years raise is 10%

Display appropriate messages. And add error handling

Batch 1 Exercise 2

- Write a function to return and display the number of years of service for an employee. The function should take the hiredate as parameter.
- Also write a code to call the function

Batch 2 Exercise 1

- Product(prod_id, prod_name, qty_on_hand)
- Order(cust_id, prod_id, order_date, qty_ordered)
- Customer(cust_id, cust_name, phone, address)

Write a stored procedure to take the cust_id, prod_id and qty_ordered as input. Procedure should check if the order for a particular customer can be fulfilled and if yes then insert the new order and update the product quantity on hand. Display appropriate message if the order cannot be fulfilled.

Output parameter must have updated value of the qty_on_hand

Batch 2 Exercise 2

- Write a function to find total quantity ordered by taking cust_id and prod_id as input parameter
- Also write a code to call the function

Batch 3 Exercise 1

- Flight(flight_id,airline_id,from_loc, to_loc,depart_time, arrival_time, capacity)
- Flight_details(flight_id,dep_date, price, available_seats)
- Passenger(passenger_id, first_name, last_name, phone, email, age)
- Ticket(ticket_id, passenger_id, flight_id, depart_date, status)

Create a stored procedure to accept passenger_id, flight_id and departure date as input. Insert the booking in the ticket table. Check if there are seats available for the required booking. If availability of seats is there then update the status to ‘confirmed’ otherwise keep it to ‘waiting’. Accordingly update available seats and display appropriate messages

Batch 3 Exercise 2

- Write a function to display the status of ticket by taking passenger's first name and last name as input parameters
- Also write a code to call the function

Batch 4 Exercise 1

account(acc_no, cust_name, balance)

customer(cust_name, cust_street, cust_city)

deposit(cust_name, acc_no, amount, deposited_date)

Write a stored procedure to take cust_name, acc_no, date of deposit and amount deposited as input.

Check if the cust_name exist in the database and the customer is having an account. If yes then insert the deposit row and update the account balance. Output the account balance by passing it as output parameter. Add error handling and appropriate messages

Batch 4 Exercise 2

- Write a function to take the account number and customer_name as input and return the amount deposited in last month.
- Also write a code to call the function