Day 8

MySQL Programming

- SQL Statements:
  - DDL : CREATE, ALTER, RENAME, DESCRIBE, DROP, TRUNCATE
  - DML : INSERT, UPDATE, DELETE
  - DQL : SELECT
  - TCL : COMMIT, ROLLBACK, SAVEPOINT
  - DCL : GRANT, REVOKE
- Programming Constructs
  - if
  - if-else
  - else if
  - switch case
  - loop(do-while, while, for)
- MySQL Programming = SQL Statements + Programming constructs
- PL/SQL is concept of ORACLE which is used for database programming.
- MySQL Programming is also called as Persistent Stored Module(PSM).
- In 1992, ANSI accepted PSM.
- PSM consist of stored procedure, function, trigger,cursor etc
- PSM contains programming constructs :
  - IF, IF-ELSE, ELSEIF, CASE, WHILE, REPEAT-UNTIL, LABLED LOOP etc.
- If we want to improve query performance then we can use PSM.

Stored Procedure

- In SQL/ MySQL, procedure is a function which do not return any value.
- Stored procedure get space on server HDD. Server save it into compiled form.
- If we want create stored procedure then we should use follwing syntax:

```
DELIMITER $$
CREATE PROCEDURE sp_name (params type)
BEGIN
    //SQL Statement1;
    //SQL Statement2;
END $$
DELIMITER ;
```

```
DELIMITER $$
CREATE PROCEDURE sp_insert_emp( )
BEGIN
    INSERT INTO emp (empno, ename, sal, deptno )
    VALUES( 7935,'ABC',5000, 30);
```

```
END $$
DELIMITER ;
```

- List procedure from database.

```
SHOW PROCEDURE STATUS;

pager less -SFX;
SHOW PROCEDURE STATUS WHERE db='classwork';

SHOW PROCEDURE STATUS LIKE 'sp%';
```

- Drop stored procedure

```
DROP PROCEDURE sp_proc1;
DROP PROCEDURE dac_db.sp_test;
```

- Call stored procedure from terminal /command line.

```
CALL sp_insert_emp( );
SELECT * FROM emp;
```

- First Create .sql File and type following program inside it.

```
DELIMITER $$
CREATE PROCEDURE sp_delete_emp( )
BEGIN
    DELETE FROM emp WHERE empno=7935;
END $$
DELIMITER ;
```

- Include and compile .sql file from terminal

```
-- SOURCE pathname.sql
SOURCE ../proc_1.sql
```

- Call Stored Procedure from terminal

```
CALL sp_delete_emp( );
```

PSM User Defined Variables

- Declare number variable in C.

```
int main( void )
{
    int number;
    return 0;
}
```

- Declare number variable in MySQL.

```
DELIMITER $$
CREATE PROCEDURE sp_proc1( )
BEGIN
    DECLARE number INT;
    DECLARE value INTEGER;
END $$
DELIMITER ;
```

- Initialize number in C

```
int main( void )
{
    int number = 10;
    return 0;
}
```

- Initialize number in MySQL

```
DELIMITER $$
CREATE PROCEDURE sp_proc1( )
BEGIN
    DECLARE number INTEGER DEFAULT 10;
END $$
DELIMITER ;
```

- Assign value 20 to the variable number in C

```
int main( void )
{
    int number;
    number = 20;
    return 0;
}
```

- Assign value 20 to the variable number in MySQL

```
DELIMITER $$
CREATE PROCEDURE sp_proc1( )
BEGIN
    DECLARE number INTEGER;
    SET number = 20;
    -- SELECT 20 INTO number;
END $$
DELIMITER ;
```

```
CREATE TABLE result
(
    value INT
);
```

```
DELIMITER //
CREATE PROCEDURE sp_proc2( )
BEGIN
    DECLARE num1 INTEGER DEFAULT 10;
    DECLARE num2 INTEGER;
    DECLARE num3 INTEGER;
    SET num2 = 20;
    -- SET num3 = 30;
    SELECT 30 INTO num3;

    INSERT INTO result VALUES( num1 );
    INSERT INTO result VALUES( num2 );
    INSERT INTO result VALUES( num3 );
END //
DELIMITER ;
```

- Check whether year is leap year or not in C

```
int main( void )
{
    int year = 2020;
    int days = 28;
    if( year % 4 == 0 )
    {
        days = 29;
    }
    return 0;
}
```

- IF : The IF statement for stored programs implements a basic conditional construct.
- Syntax

```
    IF search_condition THEN
        statement_list
    END IF
```

- Check whether year is leap year or not in MySQL

```
DELIMITER $$
CREATE PROCEDURE sp_proc3( )
BEGIN
    DECLARE year INTEGER DEFAULT 2020;
    DECLARE days INTEGER DEFAULT 28;
    IF year mod 4 = 0 THEN
        SET days = 29;
    END IF;
    INSERT INTO result VALUES ( days );
END $$
DELIMITER ;
```

```
CALL sp_proc3( );
SELECT * from result;
```

- The IF statement can have THEN, ELSE, and ELSEIF clauses, and it is terminated with END IF.

```
IF search_condition THEN
    statement_list
ELSE
    statement_list
END IF
```

```
int main( void )
{
    int year = 2020;
    int days = 0;
    if( year % 4 == 0)
        days = 29;
    else
        days = 28;
    return 0;
}
```

```
DELIMITER $$
DROP PROCEDURE IF EXISTS sp_proc4;
CREATE PROCEDURE sp_proc4( )
BEGIN
    DECLARE year INTEGER DEFAULT 2021;
    DECLARE days INTEGER DEFAULT 0;
    IF year mod 4 = 0 THEN
        SET days = 29;
    ELSE
        SET days = 28;
    END IF;
    INSERT INTO result VALUES( days );
END $$
DELIMITER ;
```

- Else if construct

```
int main( void )
{
    int month = 0;
    int days = 0;
    if( month == 1 )
        days = 31;
    else if( month == 2 )
        days = 28;
    else if( month == 3 )
        days = 31;
    else if( month == 4 )
        days = 30;
    else
        days = 0;
    return 0;
}
```

```
DELIMITER $$
DROP PROCEDURE IF EXISTS sp_proc4;
CREATE PROCEDURE sp_proc4( )
BEGIN
    DECLARE month INTEGER DEFAULT 4;
    DECLARE days INTEGER DEFAULT 0;
    IF month = 1 THEN
        SET days = 31;
    ELSEIF month = 2 THEN
        SET days = 28;
    ELSEIF month = 3 THEN
        SET days = 31;
    ELSEIF month = 4 THEN
        SET days = 30;
    ELSEIF month = 12 THEN
        SET days = 31;
    ELSE
        SET days = 0;
    END IF;
    INSERT INTO result VALUES( days );
END $$
DELIMITER ;
```

```
DELIMITER $$
DROP PROCEDURE IF EXISTS sp_proc4;
CREATE PROCEDURE sp_proc4( )
BEGIN
    DECLARE month INTEGER DEFAULT 3;
    DECLARE days INTEGER DEFAULT 0;
    IF month = 1 OR month = 3 OR month = 5 THEN
        SET days = 31;
    ELSEIF month = 2 THEN
        SET days = 28;
    ELSE
        SET days = 30;
    END IF;
    INSERT INTO result VALUES( days );
END $$
DELIMITER ;
```

- Switch Case

```
int main( void )
{
    int month = 1;
    int days = 0;
    switch( month )
    {
    case 1:
        days = 31;
        break;
    case 2:
        days = 28;
        break;
    case 3:
        days = 31;
        break;
    case 4:
        days = 30;
```

```
            break;
        default:
            days = 0;
            break;
        }
        return 0;
    }
```

- Case statement in MySQL
- Syntax

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list]
    ...
    [ELSE statement_list]
END CASE
```

```
DELIMITER $$
DROP PROCEDURE IF EXISTS sp_proc5;
CREATE PROCEDURE sp_proc5( )
BEGIN
    DECLARE month INTEGER DEFAULT 3;
    DECLARE days INTEGER DEFAULT 0;
    CASE month
    WHEN 1 THEN SET days = 31;
    WHEN 2 THEN SET days = 28;
    WHEN 3 THEN SET days = 31;
    WHEN 4 THEN SET days = 30;
    ELSE SET days = 0;
    END CASE;
    INSERT INTO result VALUES( days );
END $$
DELIMITER ;
```

- while loop in C

```
int main( void )
{
    int count = 1;
    int res = 0;
    while( count <= 10 )
    {
        res = res + count;
        count = count + 1;
    }
    //print res;
    return 0;
}
```

- While loop in MySQL
- Syntax

```
WHILE search_condition DO
    statement_list
END WHILE;
```

```
DELIMITER $$
DROP PROCEDURE IF EXISTS sp_proc6;
CREATE PROCEDURE sp_proc6( )
BEGIN
    DECLARE count INTEGER DEFAULT 1;
    DECLARE res INTEGER DEFAULT 0;
    WHILE count <= 10 DO
        SET res = res + count;
        SET count = count + 1;
    END WHILE;
    INSERT INTO result VALUES( res );
END $$
DELIMITER ;
```

- do-while construct in C

```
int main( void )
{
    int count = 1;
    int res = 0;
    do
    {
        res = res + count;
        count = count + 1;
    }while( count <= 10 );
    //print res;
    return 0;
}
```

- REPEAT-UNTIL construct in MySQL

```
REPEAT
    statement_list
UNTIL search_condition
END REPEAT
```

```
DELIMITER $$
DROP PROCEDURE IF EXISTS sp_proc6;
CREATE PROCEDURE sp_proc6( )
BEGIN
    DECLARE count INTEGER DEFAULT 1;
    DECLARE res INTEGER DEFAULT 0;
    REPEAT
        SET res = res + count;
        SET count = count + 1;
        UNTIL count > 10
    END REPEAT;
    INSERT INTO result VALUES( res );
END $$
DELIMITER ;
```

- Infinite loop in C

```
int main( void )
{
    int count = 0;
    while( 1 )
    {
        count = count + 1;
        if( count == 10 )
        {
            //print count;
            break;
        }
    }
    return 0;
}
```

- Labeled loop in MySQL
- Syntax:

```
label: LOOP
    statement_list
END LOOP label;
```

```
DELIMITER $$
DROP PROCEDURE IF EXISTS sp_proc7;
CREATE PROCEDURE sp_proc7( )
BEGIN
    DECLARE count INTEGER DEFAULT 0;
    DECLARE res INTEGER DEFAULT 0;
    label:LOOP
        SET res = res + count;
        SET count = count + 1;
        IF count = 10 THEN
            LEAVE label;
        END IF;
    END LOOP label;
    INSERT INTO result VALUES( res );
END $$
DELIMITER ;
```

- We can pass argument to the stored procedure. To catch it must specify parameters to stored procedure.
- Parameter can be:
    1. IN parameter
    2. OUT parameter
    3. INOUT parameter
- Each parameter is an IN parameter by default.

```
DELIMITER $$
DROP PROCEDURE IF EXISTS sp_insert_emp;
CREATE PROCEDURE sp_insert_emp( IN empno INTEGER, IN ename VARCHAR(50), IN
sal FLOAT, IN deptno INTEGER)
BEGIN
    INSERT INTO emp
    (empno, ename, sal, deptno)
    VALUES(empno, ename, sal, deptno);
END $$
DELIMITER ;
```

```
 SET @empno=7951;
 SET @ename='Sandeep';
 SET @sal=7500.50;
 SET @deptno=10;

 SELECT @empno, @ename, @sal, @deptno FROM DUAL;

 CALL sp_insert_emp( @empno, @ename, @sal, @deptno );
```

- OUT Parameter

```
CREATE TABLE accounts
(
    number INTEGER,
    name VARCHAR(50),
    balance FLOAT,
    type VARCHAR(50)
);
INSERT INTO accounts
VALUES( 101, 'Ketan', 75000, 'Saving');

INSERT INTO accounts
VALUES( 101, 'Akash', 30000, 'Current');
```

- Transfer fund

```
DELIMITER $$
CREATE PROCEDURE sp_transfer_fund
( IN srcNumber INTEGER,
IN destNumber INTEGER,
IN amount FLOAT,
OUT srcBalance FLOAT,
OUT destBalance FLOAT )
BEGIN
    UPDATE accounts SET balance = balance - amount WHERE number=srcNumber;

    UPDATE accounts SET balance = balance + amount WHERE
number=destNumber;

    SELECT balance INTO srcBalance FROM accounts WHERE number = srcNumber;

    SELECT balance INTO destBalance FROM accounts WHERE number =
destNumber;
END $$
DELIMITER ;
```

```
SET @srcAccNumber = 101;
SET @destAccNumber = 102;
SET @amount = 15000;

CALL sp_transfer_fund( @srcAccNumber, @destAccNumber, @amount,
@srcBalance, @destBalance );

SELECT @srcBalance "Source Bal." FROM DUAL;
SELECT @destBalance "Dest Bal." FROM DUAL;
```

- INOUT Parameter

```
DELIMITER $$
CREATE PROCEDURE sp_proc( INOUT str VARCHAR(50) )
BEGIN
   SET str=CONCAT('Hello',' ', UPPER(str));
```

```
END $$
DELIMITER ;
```

```
SET @name = 'dac';
CALL sp_proc( @name );
SELECT @name FROM DUAL;
```

- We can call stored procedure from another stored procedure.

```
DELIMITER $$
CREATE PROCEDURE sp_test( OUT str VARCHAR(50) )
BEGIN
    DECLARE name VARCHAR( 50 );
    SET name = 'sunbeam';
    CALL sp_proc( name );
    SET str = name;
END $$
DELIMITER ;
```

- Runtime error is also called as exception.
- During execution of stored procedure and function we may get runtime error/exeception. To handle it we should define error handler in PSM.
- If we want to handle error/exception in PSM then we should DECLARE HANDLER.
- Syntax:

```
DECLARE handler_action HANDLER
FOR condition_value statement;
```

- handler_action:
    - CONTINUE: Execution of the current program continues.
    - EXIT: Execution terminates.
    - UNDO: Not supported.
- Condition_value for DECLARE ... HANDLER indicates the specific condition:
    1. mysql_error_code

```
DECLARE CONTINUE HANDLER FOR 1051
BEGIN
-- body of handler
END;
```

```
2. SQLSTATE
```

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
BEGIN
-- body of handler
END;
```

```
3. condition_name
4. SQLWARNING
5. NOT FOUND
```

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
BEGIN
-- body of handler
END;
```

```
6. SQLEXCEPTION
```

```
INSERT into accounts VALUES( 102,'Rahul',50000,'Loan');
```

```
DELIMITER $$
CREATE PROCEDURE sp_insert_account(number INT, name VARCHAR(50), balance
FLOAT, type VARCHAR(50) )
BEGIN
    DECLARE EXIT HANDLER FOR 1062
    BEGIN
        SELECT 'Duplicate Account';
    END;
    INSERT INTO accounts VALUES( number, name, balance, type );
    -- stm1
    -- stm2
    -- stm3

END $$
DELIMITER ;
```

```
DELIMITER $$
CREATE PROCEDURE sp_insert_account(number INT, name VARCHAR(50), balance
FLOAT, type VARCHAR(50) )
```

```
BEGIN
    DECLARE EXIT HANDLER FOR 1062 SELECT 'Duplicate Account';
    INSERT INTO accounts VALUES( number, name, balance, type );
    -- stm1
    -- stm2
    -- stm3

END $$
DELIMITER ;
```

- The DECLARE ... CONDITION statement declares a named error condition, associating a name with a condition that needs specific handling.
- Syntax:

```
DECLARE condition_name CONDITION FOR condition_value
```

```
DECLARE duplicate_entry CONDITION FOR 1062;
```

```
DELIMITER $$
CREATE PROCEDURE sp_insert_account(number INT, name VARCHAR(50), balance
FLOAT, type VARCHAR(50) )
BEGIN
    DECLARE duplicate_entry CONDITION FOR 1062;
    DECLARE CONTINUE HANDLER FOR duplicate_entry SELECT 'Duplicate
Account';
    INSERT INTO accounts VALUES( number, name, balance, type );
END $$
DELIMITER ;
```

```
DELIMITER $$
CREATE PROCEDURE sp_insert_account(number INT, name VARCHAR(50), balance
FLOAT, type VARCHAR(50) )
BEGIN
    DECLARE duplicate_entry CONDITION FOR SQLSTATE '23000';
    DECLARE CONTINUE HANDLER FOR duplicate_entry
    BEGIN
        SELECT 'Duplicate Account';
    END;

    INSERT INTO accounts VALUES( number, name, balance, type );
END $$
DELIMITER ;
```