Department of Computer Science & Engineering,
CS246 – Database Management Systems Lab

Stored Procedures, Stored Functions

Overview

- Session
- Types of Variables
- Blocks
- Nested Blocks
- Conditional statements
- Loops
- Stored functions
- Stored procedures

Session

Session - 01

- A session refers to a period of interaction between a client application and the MySQL server
- When a client application connects to the MySQL server, a session is established, allowing the client to execute queries and perform various database operations.
- Connection: A session begins when a client application establishes a connection to the MySQL server. The session ends when the connection is Terminated.
- Duration: The duration of a session depends on the connection settings and the actions taken by the client application. Sessions can last from a fraction of a second to several hours or more, depending on the requirements of the application

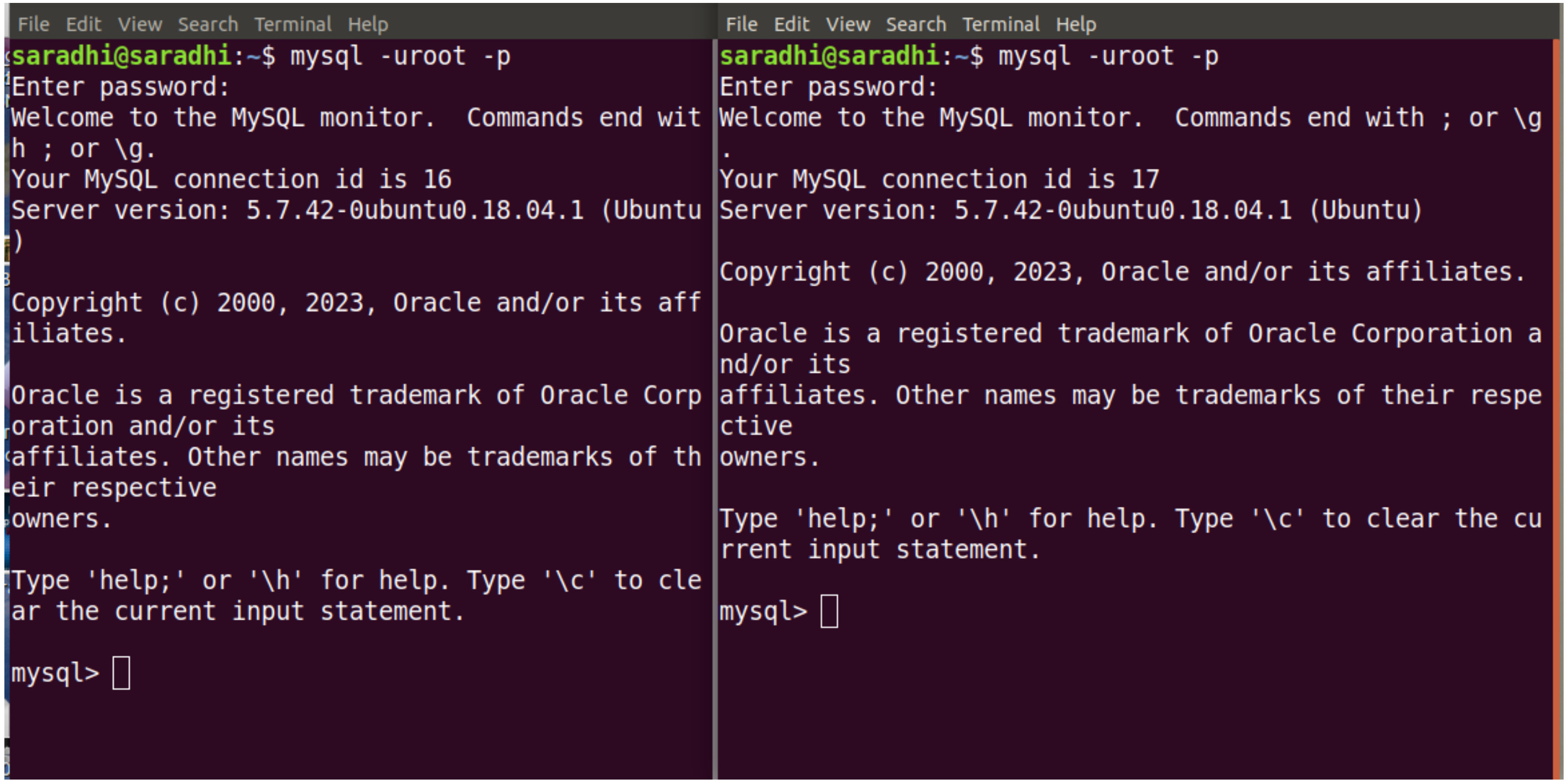
Session - 02

- Scope: Each session is isolated from other sessions. Changes made within one session (e.g., variable assignments, temporary tables) do not affect other sessions.

- Session Variables: Session variables are variables that exist for the duration of the session. They are prefixed with `@@session.` and can be used to control various aspects of the session behavior (e.g., setting auto-increment values, defining time zone settings)

- Sessions play a crucial role in the interaction between client applications and the MySQL server

Session - 03



```
saradhi@saradhi:~$ mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end wit
h ; or \g.
Your MySQL connection id is 16
Server version: 5.7.42-0ubuntu0.18.04.1 (Ubuntu
)

Copyright (c) 2000, 2023, Oracle and/or its aff
iliates.

Oracle is a registered trademark of Oracle Corp
oration and/or its
affiliates. Other names may be trademarks of th
eir respective
owners.

Type 'help;' or '\h' for help. Type '\c' to cle
ar the current input statement.

mysql> 
```

```
saradhi@saradhi:~$ mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g
.
Your MySQL connection id is 17
Server version: 5.7.42-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation a
nd/or its
affiliates. Other names may be trademarks of their respe
ctive
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the cu
rrent input statement.

mysql> 
```

Types of Variables

- User defined variables
- Parameters & local variables
- System variables
- Global variables

User defined variables

User defined variables - 01

- A user-defined variable in MySQL is a variable that is created and managed by the user within a session
- It allows you to store values temporarily and reference them in subsequent queries or statements within the same session.
- User-defined variables are preceded by an '@' symbol.

```
SET @var1 = 10;
SET @hex_1 = X'41'

SELECT @var1;

SELECT @hex1;
```

```
SET @v1 = 10;
SET @v2 = 'CS246';
SET @v3 = 8.94;

SELECT @v1, @v2, @v3;
```

User defined variables - 02

- User variables are intended to provide data values.
- They cannot be used directly in an SQL statement as an identifier or as part of an identifier, such as in contexts where a table or database name is expected

```
SET @column1 = 'roll_number';

SELECT roll_number FROM student;

SELECT @column1 FROM student;

SELECT `@column1` FROM student;
```

User defined variables - 03

- **SQL statements as strings and executing them**

```
SET @column1 = 'roll_number';

SET @stmt_1 = CONCAT("SELECT ", @column1, "FROM student");

PREPARE stmt FROM @stmt_1;

EXECUTE stmt;

DEALLOCATE PREPARE stmt;
```

Local variables

Local variables - 01

- A local variable is a variable that is declared within the scope of a stored program (such as a stored procedure, function, or trigger) and exists only within that scope.
- Local variables are used to store values temporarily and can be referenced within the stored program in which they are defined.
- Declaring local variables

```
DECLARE v1 INT;
DECLARE v2 FLOAT;
DECLARE v3 CHAR(20);
DECLARE v4 VARCHAR(30);
DECLARE v5 DECIMAL(10, 2);
```

Initializing local variables

```
SET v1 = 10;
SET v2 = 7.33;
SET v3 = 'ATUL';
SET v4 = 'DBMS Lab';
SET v5 = 8.94;
```

Local variables - 02

- Initialization at the time of declaration

```
DECLARE v1 INT DEFAULT 0;
DECLARE v2 FLOAT DEFAULT 0.0;
DECLARE v3 CHAR(20) DEFAULT 'NAME';
DECLARE v4 VARCHAR(30) DEFAULT 'Course Title';
DECLARE v5 DECIMAL(10, 2) DEFAULT 0.00;
```

```
SET v1 = 10;
SET v2 = 7.33;
SET v3 = 'ATUL';
SET v4 = 'DBMS Lab';
SET v5 = 8.94;
```

Local variables - 03

- **Scope of local variable The scope of a local variable is the BEGIN END block within which it is declared.**

```
BEGIN

   DECLARE v1 INT DEFAULT 0;
   DECLARE v2 FLOAT DEFAULT 0.0;
   DECLARE v3 CHAR(20) DEFAULT 'NAME';
   DECLARE v4 VARCHAR(30) DEFAULT 'Course Title';
   DECLARE v5 DECIMAL(10, 2) DEFAULT 0.00;

END
```

Local variables - 04

- Local variable in SQL statements and their interpretation
- xname is NOT a COLUMN name in table1.
- SELECT statement WILL NOT interpret xname as COLUMN name

```
BEGIN

  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;

  SELECT xname,   id
  INTO   newname, xid
  FROM   table1
  WHERE  xname = xname;

  SELECT newname;

END
```

System variables

System variables - 01

- System variables are global variables that control various aspects of the server's behavior.
- These variables affect the operation of the MySQL server itself, rather than specific sessions or stored programs.
- System variables can be used to configure settings related to performance, behavior, and resource utilization.

System variables - 02

- **Viewing ALL the system variables**

```
SHOW VARIABLES;
```

- **Viewing specific system variable**

```
SELECT @@max_connections;
```

System variables - 03

- Creating a custom system variable

- Custom system variables can be created by modifying the MySQL configuration file (my.cnf or my.ini) and adding a new entry under the [mysqld] section.

```
[mysqld]
login_retries = 10;
```

- Save the file and restart mysql server as

```
sudo service restart mysql
```

System variables - 04

- Creating a custom system variable whose scope is a session

```
SET @@session.variable_name = value;
```

- Example:

```
SET @@session.sql_mode = 'STRICT_TRANS_TABLES';
```

- Keep in mind that **not all system variables can be modified at the session level.** Some system variables may be read-only or may require specific privileges to be modified.

Global variables

Global variables - 01

- Global variables that control various aspects of the server's behavior.
- Global variables can be used to control various aspects of the MySQL server's behavior, such as setting resource limits, adjusting performance parameters, or configuring server options.
- They provide a way to customize the behavior of the MySQL server to suit specific requirements or preferences.
- They are created in an identical way as that of system variables.

Stored Procedures

Stored procedures - 01

- A stored procedure is a set of SQL statements that are stored on the server and can be executed repeatedly without having to reissue the individual statements each time.
- Stored procedure consists of one or more blocks
- Each block is of structure BEGIN …. END

```
DELIMITER //

CREATE PROCEDURE p1()
BEGIN

END //

DELIMITER ;
```

Stored procedures - 02

- Various types of declarations can appear in a block
- Order of declaration of these are very important. Each declaration must occur as specified below
- All variables must be declared at the beginning of a block
- All cursors must be declared after variable declaration
- All error handlers are declared after cursors
- Program code then can start

Stored procedures - 03

- Various types of declarations can appear in a block
- Order of declaration of these are very important. Each declaration must occur as specified below
- All variables must be declared at the beginning of a block
- All cursors must be declared after variable declaration
- All error handlers are declared after cursors
- Program code then can start

```
DELIMITER //
CREATE procedure p1()
BEGIN
  -- Variable declaration
  DECLARE v1 INT;
  DECLARE v2 CHAR(20);

  --Cursor declaration

  --Error handler declaration

  --Actual program begins
END//

DELIMITER ;
```

Stored procedures - 04

- Creating stored procedure
- Calling stored procedure
- Deleting stored procedure

Stored procedures - 05

```
DELIMITER //

CREATE procedure p1()
BEGIN
  -- Variable declaration
  DECLARE v1 INT DEFAULT 10

  BEGIN
    DECLARE v2 INT DEFAULT 20;
    SET v2=30;
    SELECT v2;
  END;
  SET v1 = 20;
  SELECT v1;
END //
DELIMITER ;
```

Stored procedures - 06

```
MySQL > source p1.sql

MySQL > call p1();

MySQL > DROP procedure p1;
```

Stored procedures - 07

```
CREATE TABLE employees(eid int, fname char(10), lname char(10),
                       salary DECIMAL(10, 2), primary key(eid));

CREATE procedure p2(in employee_id int)
BEGIN
  SELECT * from employees where eid=employee_id
END //
DELIMITER ;
```

- employee_id is a parameter to procedure
- The keyword IN specifies that employee_id is a read only variable to p2. That is changes to employee_id within p2 will not affect outside the scope of p2
- Call p2(703); -- Retrieve details of employee id 703

Stored procedures - 08

- Takes three types of arguments/parameters
- IN - An IN parameter passes a value into a procedure. The procedure might modify the value, but the modification is not visible to the caller when the procedure returns.
- OUT - An OUT parameter passes a value from the procedure back to the caller.
- INOUT - An INOUT parameter is initialized by the caller, can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.

Stored procedures – 09 - IN

```
MySQL > SET @eid=703;

MySQL > source p3.sql

MySQL > call p3(@eid);

MySQL > SELECT @eid // 703

MySQL > DROP procedure p3;
```

```
DELIMITER //

CREATE PROCEDURE p3(in emp_id int)
BEGIN
  SELECT emp_id;
  SET emp_id = 713;
END //

DELIMITER ;
```

Stored procedures – 10 - OUT

```
MySQL > source p4.sql

MySQL > call p4(@v1);

MySQL > SELECT @v1 // 713

MySQL > DROP procedure p4;
```

```
DELIMITER //

CREATE PROCEDURE p4(out meid int)
BEGIN
  SET mid = 713;
END //

DELIMITER ;
```

Stored procedures – 10 - INOUT

```
MySQL > SET @eid=703;

MySQL > source p5.sql

MySQL > call p5(@eid);

MySQL > SELECT @eid // 713

MySQL > DROP procedure p5;
```

```
DELIMITER //

CREATE PROCEDURE p5(inout emp_id int)
BEGIN
  SELECT emp_id;
  SET emp_id = 713;
END //

DELIMITER ;
```

Stored procedures – 11

```
DELIMITER //

CREATE PROCEDURE p6()
BEGIN
  DECLARE v1 int DEFAULT 10;
  BEGIN
    DECLARE v2 int DEFAULT 20;
    SET v2=25;
  END;

  SET v1 = 15;

END //

DELIMITER ;
```

Stored procedures – 13

```
DELIMITER //

CREATE PROCEDURE p7()
BEGIN
  DECLARE v1 int DEFAULT 10;
  BEGIN
    DECLARE v2 int DEFAULT 20;
    SET v2=25;
  END;

  SET v1 = 15;
  SELECT v1, v2, 'This statement causes an error';

END //

DELIMITER ;
```

Stored procedures – 14

```
DELIMITER //

CREATE PROCEDURE p8()
BEGIN
  DECLARE v1 int DEFAULT 10;
  BEGIN
    DECLARE v2 int DEFAULT 20;
    SET v2=25;
    SET v1=100; -- over-writing of v1 occurs
  END;

  SELECT v1;
END //

DELIMITER ;
```

Conditional statements

Conditional statements - 01

```
DELIMITER //

CREATE PROCEDURE p9(in sales_id int, in sales_value float)
BEGIN
  IF( sales_value > 200 )
  THEN
    CALL apply_free_shipping(sales_id);

    IF( sales_value > 500 )
    THEN
      Call apply_discount(sales_id, 20);
    END IF;
  END IF;

END //

DELIMITER ;
```

Conditional statements - 02

```
DELIMITER //

CREATE PROCEDURE p10(in cpi float)
BEGIN
  IF( cpi > 7.0 )
  THEN
    SELECT roll, name from student where deot = 'EEE';
  ELSE IF( cpi between 5.0 and 7.0 )
    SELECT roll, name from student where dept = 'BSBE';
  ELSE
    SELECT roll, name from student where dept <> 'EEE' AND dept <> 'BSBE';
  END IF;
END //

DELIMITER ;
```

Conditional statements - 03

```
DELIMITER //

CREATE PROCEDURE p11(in sale_value float, in customer_status ENUM(PLATINUM,
GLOD, SILVER, BRONZE) , in sale_id int)
BEGIN
  DECLARE dummy int DEFAULT -1;
  CASE
    WHEN( sale_value > 200 and customer_status = PLATINUM )
    THEN
      CALL apply_discount(sale_id, 20);
    WHEN( sale_value > 200 and customer_status = GOLD )
    THEN
      CALL apply_discount(sale_id, 15);
    ELSE
      dummy = 10;
  END CASE;
END //

DELIMITER ;
```

Loops

Loops - 01

```
DELIMITER //

CREATE PROCEDURE p12()
BEGIN
  DECLARE a int default 1;

  Myloop: LOOP
    SET a = a + 1;
    IF( a = 10 )
    THEN
      Leave Myloop;
     END IF;
  END LOOP Myloop;
  SELECT 'I can count upto 10';
END //

DELIMITER ;
```

Loops - 02

```
DELIMITER //

CREATE PROCEDURE p13()
BEGIN
  DECLARE a int default 1;

  Myloop: REPEAT
    SET a = a + 1;
    IF( MOD(a, 2) = 1 )
    THEN
      SELECT CONCAT(a, ' is odd');
    END IF;
    UNTIL a >= 10
  END REPEAT;
END //

DELIMITER ;
```

Loops - 03

```
DELIMITER //

CREATE PROCEDURE p14()
BEGIN
  DECLARE a int default 1;

  Myloop: WHILE a <= 10 DO

    IF( MOD(a, 2) = 1 )
    THEN
      SELECT CONCAT(a, ' is odd');
    END IF;
    SET a = a + 1;
  END WHILE Myloop;
END //

DELIMITER ;
```

Loops - 04

```
DELIMITER //

CREATE PROCEDURE p15()
BEGIN
  DECLARE a int default 1;
  DECLARE b int default 1;
  Loop01: LOOP
    SET b = 1;
    LOOP02: LOOP
      SELECT CONCAT(a, ' times ', b, ' is ', a * b);
      SET b = b + 1;
      IF ( b > 10 )
      THEN
        LEAVE LOOP02;
      END IF;
    END LOOP LOOP02;
    SET a = a + 1;
    IF( a > 10 ) THEN LEAVE LOOP01; END IF;
  END LOOP LOOP01;
END //

DELIMITER ;
```

Stored procedure examples

Stored procedures - 04

```
DELIMITER //

CREATE PROCEDURE p16()
BEGIN
  DECLARE a int default 1;
  DROP TABLE IF EXISTS test_table;

  CREATE TABLE test_table(id int, some_data char(10), primary key(id));

  WHILE( a <= 10 )
  DO
    INSERT INTO test_table(id, some_date) values (a, CONCAT("record ", a));
    Set a = a + 1;
  END WHILE;

END //

DELIMITER ;
```

Stored functions

Stored functions - 01

- Similar to stored procedure declaration
- A function must have a return value
- A procedure SHOULD NOT have a return value
- Stored functions can be called in select or other SQL statements
- Example

Functions examples

```
SELECT roll_number,
       CONCAT(sur_name, ' ', first_name, ' ' last_name) as full_name
FROM   Student
WHERE  Dept = 'EEE'
```

```
SELECT roll_number,
       ABS(quiz1_marks)
FROM   Student
WHERE  Dept = 'EEE'
```

```
SELECT roll_number, ROUND(SPI, 2) , ROUND((CPI , 2)
FROM   Student
WHERE  Dept = 'EEE'
```

Functions examples

```
SELECT roll_number, DAYNAME(held_on)
FROM   Attendance
WHERE  cid = 'cs246'
```

```
SELECT DATE_ADD('2024-04-01', INTERVAL 1 DAY);

SELECT DATE_SUB('2024-04-01', INTERVAL 1 YEAR);

SELECT DATE_ADD('2024-04-01 09:17:24', INTERVAL 1 SECOND);
```

C

Thank You!