ABHINAV PATEL
500119461
R2142230047
B.TECH CSE BATCH 2
DBMS LAB

EXPERIMENT - 12

Title: To understand the concepts of Sequence.
Objective: Students will be able to implement the concept of sequence.
1) Create a sequence by name EMPID_SEQ starting with value 100 with an interval of 1.

```
3
4    -- 1) Create a table that simulates the EMPID_SEQ sequence (since MySQL doesn't support CREATE SEQUENCE)
5    CREATE TABLE EMPID_SEQ (
6        seq_value INT NOT NULL AUTO_INCREMENT PRIMARY KEY
7    );
8
9    -- Insert a starting value (this is equivalent to creating a sequence starting with 100)
10   INSERT INTO EMPID_SEQ (seq_value) VALUES (100);
11
```

Output

Action Output

#	Time	Action	Message
1	21:56:21	CREATE TABLE EMPID_SEQ (seq_value INT NOT NULL AUTO_INCREMENT PRIMARY KEY)	0 row(s) affected
2	21:56:21	INSERT INTO EMPID_SEQ (seq_value) VALUES (100)	1 row(s) affected

2) Write a SQL command for finding the current and the next status of EMPID_SEQ.

```
11
12   -- 2) SQL command to find the current and next status of the simulated sequence
13   -- Current value (last inserted value)
14   SELECT seq_value FROM EMPID_SEQ ORDER BY seq_value DESC LIMIT 1;
15
16   -- Next value (simulate getting the next sequence value)
17   INSERT INTO EMPID_SEQ (seq_value) VALUES (NULL);
18   SELECT seq_value FROM EMPID_SEQ ORDER BY seq_value DESC LIMIT 1;
19
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

seq_value
101
NULL

EMPID_SEQ 1 EMPID_SEQ 2

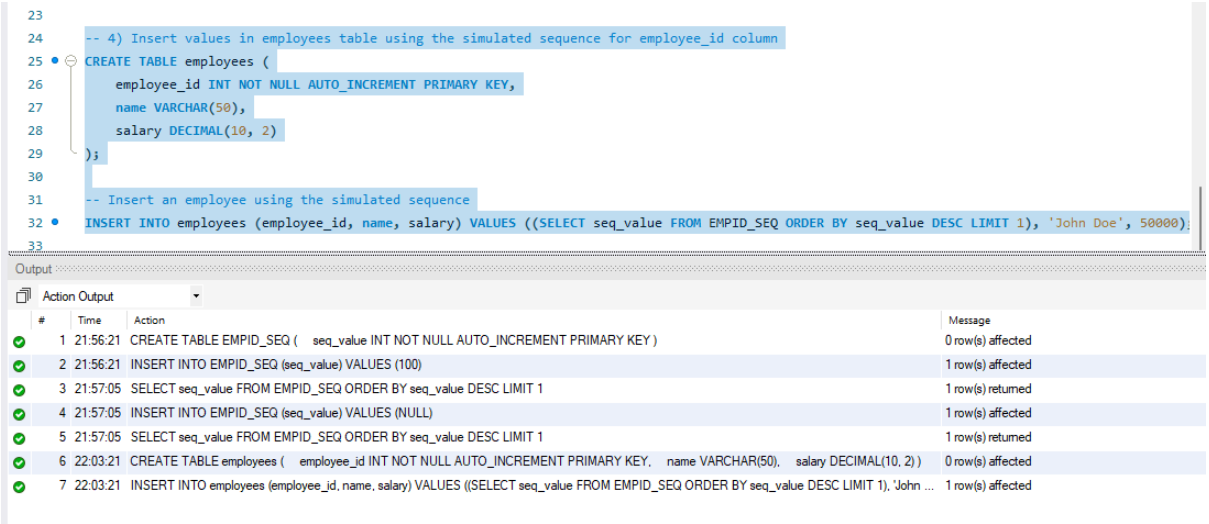
Output

Action Output

#	Time	Action	Message
1	21:56:21	CREATE TABLE EMPID_SEQ (seq_value INT NOT NULL AUTO_INCREMENT PRIMARY KEY)	0 row(s) affected
2	21:56:21	INSERT INTO EMPID_SEQ (seq_value) VALUES (100)	1 row(s) affected
3	21:57:05	SELECT seq_value FROM EMPID_SEQ ORDER BY seq_value DESC LIMIT 1	1 row(s) returned
4	21:57:05	INSERT INTO EMPID_SEQ (seq_value) VALUES (NULL)	1 row(s) affected
5	21:57:05	SELECT seq_value FROM EMPID_SEQ ORDER BY seq_value DESC LIMIT 1	1 row(s) returned

3) Change the Cache value of the sequence EMPID_SEQ to 20 and maxvalue to 1000.
Ans3) MySQL does not have a direct "cache" or "maxvalue" option like Oracle sequences. We simulate this behaviour manually if needed.

4) Insert values in employees table using sequences for employee_id column.



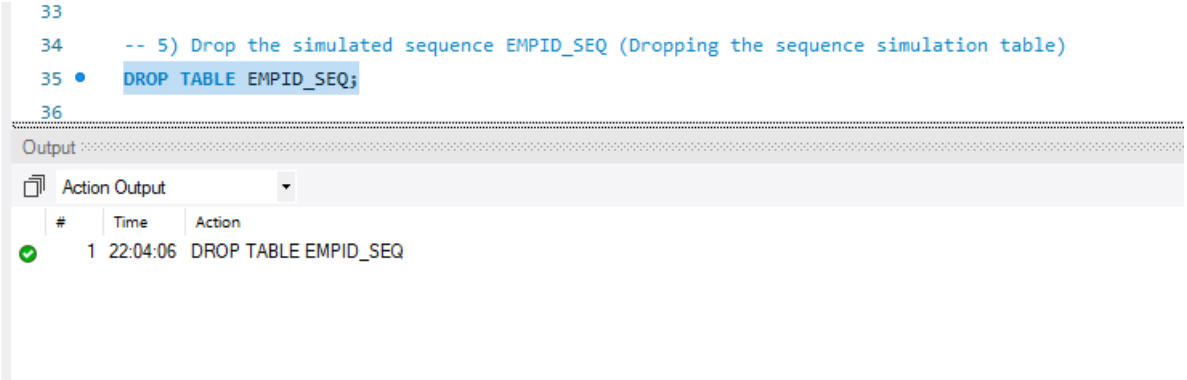
```
23
24    -- 4) Insert values in employees table using the simulated sequence for employee_id column
25    CREATE TABLE employees (
26        employee_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
27        name VARCHAR(50),
28        salary DECIMAL(10, 2)
29    );
30
31    -- Insert an employee using the simulated sequence
32    INSERT INTO employees (employee_id, name, salary) VALUES ((SELECT seq_value FROM EMPID_SEQ ORDER BY seq_value DESC LIMIT 1), 'John Doe', 50000);
33
```

Output

Action Output

#	Time	Action	Message
1	21:56:21	CREATE TABLE EMPID_SEQ (seq_value INT NOT NULL AUTO_INCREMENT PRIMARY KEY)	0 row(s) affected
2	21:56:21	INSERT INTO EMPID_SEQ (seq_value) VALUES (100)	1 row(s) affected
3	21:57:05	SELECT seq_value FROM EMPID_SEQ ORDER BY seq_value DESC LIMIT 1	1 row(s) returned
4	21:57:05	INSERT INTO EMPID_SEQ (seq_value) VALUES (NULL)	1 row(s) affected
5	21:57:05	SELECT seq_value FROM EMPID_SEQ ORDER BY seq_value DESC LIMIT 1	1 row(s) returned
6	22:03:21	CREATE TABLE employees (employee_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY, name VARCHAR(50), salary DECIMAL(10, 2))	0 row(s) affected
7	22:03:21	INSERT INTO employees (employee_id, name, salary) VALUES ((SELECT seq_value FROM EMPID_SEQ ORDER BY seq_value DESC LIMIT 1), 'John ...	1 row(s) affected

5) Drop sequence EMPID_SEQ.



```
33
34    -- 5) Drop the simulated sequence EMPID_SEQ (Dropping the sequence simulation table)
35    DROP TABLE EMPID_SEQ;
36
```

Output

Action Output

#	Time	Action
1	22:04:06	DROP TABLE EMPID_SEQ

6) Create a sequence called REVERSE to generate numbers in the descending order from 10000 to 1000 with a decrement of 5.

```
-- 6) Create a sequence called REVERSE to generate numbers in descending order from 10000 to 1000 with a decrement of 5
-- We need a stored procedure to handle the reverse sequence insertion.

DELIMITER $$

CREATE PROCEDURE GenerateReverseSequence()
BEGIN
    DECLARE seq_value INT DEFAULT 10000;

    -- Loop to insert values from 10000 to 1000 with a decrement of 5
    WHILE seq_value >= 1000 DO
        INSERT INTO REVERSE (seq_value) VALUES (seq_value);
        SET seq_value = seq_value - 5;
    END WHILE;
END$$

DELIMITER ;

-- Now we can call the stored procedure to generate the reverse sequence.
CALL GenerateReverseSequence();
```

EXPERIMENT - 13

Title: To understand the concepts of PL/SQL programming.
Objective: Students will be able to implement the basic concepts of PI/SQL.
1) Write a PL/SQL code to accept the value of A, B & C display which is greater.

```
4   -- 1) MySQL query to accept the value of A, B & C and display which is greater
5   -- Using variables to store values
6   SET @A = 10;
7   SET @B = 20;
8   SET @C = 30;
9
10  -- Checking which value is greater
```

Output

Action Output

#	Time	Action
1	21:45:30	SET @A = 10
2	21:45:30	SET @B = 20
3	21:45:30	SET @C = 30

```
-- Checking which value is greater
SELECT
    CASE
        WHEN @A > @B AND @A > @C THEN 'A is greater'
        WHEN @B > @A AND @B > @C THEN 'B is greater'
        ELSE 'C is greater'
    END AS GreaterValue;

-- 2) MySQL query to create a simple loop that displays the message 20
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

GreaterValue
C is greater

2) Using PL/SQL Statements create a simple loop that display message “Welcome to PL/SQL Programming” 20 times.

```
-- 2) MySQL query to create a simple loop that displays the message 20 times
-- We use a stored procedure with a loop
DELIMITER $$

CREATE PROCEDURE DisplayMessage()
BEGIN
    DECLARE counter INT DEFAULT 1;
    WHILE counter <= 20 DO
        SELECT 'Welcome to PL/SQL Programming';
        SET counter = counter + 1;
    END WHILE;
END $$

DELIMITER ;

-- To execute the stored procedure:
CALL DisplayMessage();

-- 3) MySQL query to find the factorial of a number
DELIMITER $$
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Welcome to PL/SQL Programming
Welcome to PL/SQL Programming

sult 8 | Result 9 | Result 10 | Result 11 | Result 12 | Result 13 | Result 14 | Result 15 | Result 16 | Result 17 | Result 18 | Result 19 | Result 20 | Result 21 | Rea

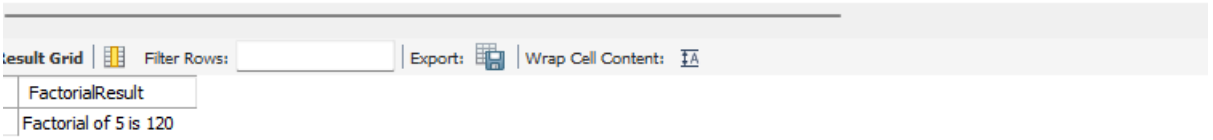
Output

Action Output

#	Time	Action	Message
14	21:48:25	-- To execute the stored procedure: CALL DisplayMessage()	1 row(s) returned
15	21:48:25	-- To execute the stored procedure: CALL DisplayMessage()	1 row(s) returned
16	21:48:25	-- To execute the stored procedure: CALL DisplayMessage()	1 row(s) returned
17	21:48:25	-- To execute the stored procedure: CALL DisplayMessage()	1 row(s) returned
18	21:48:25	-- To execute the stored procedure: CALL DisplayMessage()	1 row(s) returned
19	21:48:25	-- To execute the stored procedure: CALL DisplayMessage()	1 row(s) returned
20	21:48:25	-- To execute the stored procedure: CALL DisplayMessage()	1 row(s) returned
21	21:48:25	-- To execute the stored procedure: CALL DisplayMessage()	1 row(s) returned

3) Write a PL/SQL code block to find the factorial of a number.

```
38  -- 3) MySQL query to find the factorial of a number
39  DELIMITER $$
40
41  CREATE PROCEDURE FindFactorial(IN num INT)
42  BEGIN
43      DECLARE fact INT DEFAULT 1;
44      DECLARE i INT DEFAULT 1;
45
46      -- Loop to calculate factorial
47      WHILE i <= num DO
48          SET fact = fact * i;
49          SET i = i + 1;
50      END WHILE;
51
52      SELECT CONCAT('Factorial of ', num, ' is ', fact) AS FactorialResult;
53  END $$
54
55  DELIMITER ;
56
57  -- To execute the stored procedure for calculating factorial of 5:
58  CALL FindFactorial(5);
59
```



4) Write a PL/SQL program to generate Fibonacci series.

```
60      -- 4) MySQL query to generate Fibonacci series
61      DELIMITER $$
62
63      CREATE PROCEDURE GenerateFibonacci(IN n INT)
64      BEGIN
65          DECLARE a INT DEFAULT 0;
66          DECLARE b INT DEFAULT 1;
67          DECLARE temp INT;
68          DECLARE counter INT DEFAULT 1;
69
70          SELECT 'Fibonacci Series:';
71
72          WHILE counter <= n DO
73              SELECT a;
74              SET temp = a + b;
75              SET a = b;
76              SET b = temp;
77              SET counter = counter + 1;
78          END WHILE;
79      END $$
80
81      DELIMITER ;
82
83      -- To execute the stored procedure for generating a Fibonacci series for 10 terms:
84      CALL GenerateFibonacci(10);
85
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	a
▶	34

Result 25 | Result 26 | Result 27 | Result 28 | Result 29 | Result 30 | Result 31 | Result 32 | Result 33 | Result 34

Output

Action Output

#	Time	Action
11	21:50:13	-- To execute the stored procedure for generating a Fibonacci series for 10 terms: CALL GenerateFibonacci(10)
12	21:50:13	-- To execute the stored procedure for generating a Fibonacci series for 10 terms: CALL GenerateFibonacci(10)

5) Write a PL/SQL code to fund the sum of first N numbers

```
-- 5) MySQL query to find the sum of first N numbers
DELIMITER $$

CREATE PROCEDURE FindSumOfFirstNNumbers(IN N INT)
BEGIN
    DECLARE sum INT DEFAULT 0;
    DECLARE i INT DEFAULT 1;

    -- Loop to calculate the sum
    WHILE i <= N DO
        SET sum = sum + i;
        SET i = i + 1;
    END WHILE;
```