MySQL: Session Variables & Stored Procedures

CS 377: Database Systems

Recap: SQL

- Data definition
    - Database Creation (CREATE DATABASE)
    - Table Creation (CREATE TABLE)
- Query (SELECT)
- Data update (INSERT, DELETE, UPDATE)
- View definition (CREATE VIEW)

Session Variables

- A session starts with a connection to the SQL server and ends when the connection is closed
- Session variables can be created anytime during a SQL session
  - Exists for the remainder of the SQL session
  - Always begins with the symbol “@“ (e.g, @x, @count)
- Not part of the SQL standard - so may differ across implementations

MySQL Session Variables Syntax

- Assign a value
  - Syntax:
    SET <varName> = express;
  - Example: **SET @count = 100;**
- Assign the result of a single-valued query to a session variable
  - Syntax:
    SELECT … INTO @varname
    FROM …
    WHERE …
  - Example: **SELECT max(salary) INTO @maxSal FROM employee;**

MySQL Session Variable Syntax (2)

- Use a session variable in a query

  Example:
  ```
  SELECT fname, lname
  FROM    employee
  WHERE  salary = @maxSal;
  ```

Temporary Tables

- Store and process intermediate results using the same selection, update, and join capabilities in typical SQL tables
- Temporary tables are deleted when the current client session terminates
- Each vendor has a different syntax for creating temporary tables

MySQL Temporary Table Syntax

- Syntax:
```
CREATE TEMPORARY TABLE
…
```

- Example using a select statement:
```
CREATE TEMPORARY TABLE top5Emp
AS ( SELECT *
        FROM employee
        ORDER BY salary DESC
        LIMIT 5 );
```

- Example with empty table:
```
CREATE TEMPORARY TABLE empSum
( ssn               CHAR(9) NO NULL,
  dependentNo INT DEFAULT 0,
  salary            DECIMAL(7,2));
```

View vs Temporary Table

- View is not a real table and just a “stored” query
- Views persist beyond a session
- Temporary table disappears after session is over
- Temporary tables are useful if your query is “long” and you are accessing the results from multiple queries
- Tradeoff between processing and storage

Stored Procedures

- Generalization of SQL by adding programming language-like structure to the SQL language
- Structures typically available in stored procedure
  - Variables
  - IF statement
  - LOOP statement
- Most database vendors support them in some form

Stored Procedure Syntax

- Syntax:
```
CREATE PROCEDURE <procedure name>
(parameters)
    BEGIN
        <statements of the procedure>
    END <DELIMITER>
```

- <DELIMITER> is a special symbol used by MySQL to end a command line - default is semi-colon (;)

- A stored procedure can only be used within the database where the stored procedure was defined

Example: Stored Procedure

- Define a procedure to get the first and last name of all employees

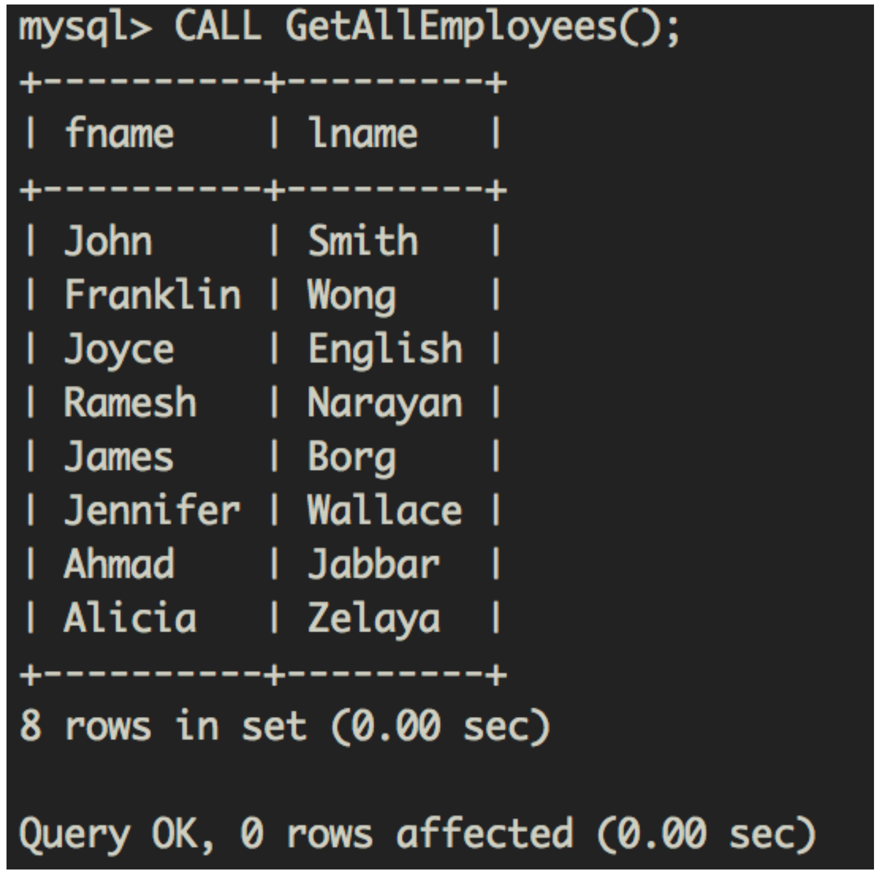
```
DELIMITER //
CREATE PROCEDURE GetAllEmployees()
    BEGIN
    SELECT fname, lname  FROM employee;
    END //
DELIMITER ;
```

To store the symbol ; inside the stored procedure, we need to redefine the delimiting symbol using the command DELIMITER //

Stored Procedure Usage

- Invoke (call) a procedure:
  CALL procedureName(parameters);

- Example:



```
mysql> CALL GetAllEmployees();
+----------+---------+
| fname    | lname   |
+----------+---------+
| John     | Smith   |
| Franklin | Wong    |
| Joyce    | English |
| Ramesh   | Narayan |
| James    | Borg    |
| Jennifer | Wallace |
| Ahmad    | Jabbar  |
| Alicia   | Zelaya  |
+----------+---------+
8 rows in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Stored Procedure Information

- Show the name of stored procedures
  - All procedures:
    SHOW PROCEDURE STATUS;
  - Only procedures with a certain name
    SHOW PROCEDURE STATUS WHERE name LIKE <pattern>;
- Get definition
  SHOW CREATE PROCEDURE <procedure name>;
- Removing procedures from system
  DROP PROCEDURE <procedure name>;

Stored Procedure Details

- A stored procedure can have any number of statements
  Example:
  ```
  DELIMITER //
  CREATE PROCEDURE GetAllEmpDepts()
      BEGIN
      SELECT fname, lname FROM employee
      SELECT dname, mgrssn FROM department;
      END
  DELIMITER ;
  ```

- A comment line is started by the symbol **--**
  Example:
  ```
  -- This is a comment line
  ```

Stored Procedures: Local Variables

- A local variable only exists within a stored procedure (similar to those in programming languages like Java or C)
- Do not use @ as a prefix to a local variable, this is always a session variable in MySQL
- Syntax:
  DECLARE <var_name> DATATYPE [DEFAULT value];

Example: Local Variable

```
DELIMITER //

CREATE PROCEDURE Variable1()
 BEGIN
 DECLARE  myvar  INT ;
 SET myvar = 1234;
 SELECT concat('myvar = ', myvar ) ;
 END //

DELIMITER ;
```

Stored Procedure: Local Variable (2)

- Similar to session variables, you can assign a value to a variable or store a query with a single value
  - Assign value:
    SET <varname> = expression;
  - Assign a result from single query
    SELECT … INTO <varname>
    FROM …
    WHERE …
- **BEGIN** and **END** keywords defines the scopes of local variables

Example: Local Variable From Query

```
DELIMITER //

CREATE PROCEDURE Variable2()
 BEGIN
 DECLARE  myvar  INT ;
 SELECT sum(salary) INTO myvar
 FROM   employee
 WHERE  dno = 4;
 SELECT CONCAT('myvar = ', myvar );
 END //

DELIMITER ;
```

Example: Local Variable Scope

```
DELIMITER //

CREATE PROCEDURE Variable3()
BEGIN
DECLARE x1 CHAR(5) DEFAULT 'outer';
SELECT x1;
  BEGIN
  --  x2 only inside inner scope !
  DECLARE x2 CHAR(5) DEFAULT 'inner';
  SELECT x1;
  SELECT x2;
  END;
SELECT x1;
END; //

DELIMITER ;
```

Example: Local Variable Shadowing

```
DELIMITER //

CREATE PROCEDURE Variable4()
BEGIN
DECLARE x1 CHAR(5) DEFAULT 'outer';
SELECT x1;
  BEGIN
  DECLARE x1 CHAR(5) DEFAULT 'inner';
  SELECT x1;
  END;
SELECT x1;
END; //

DELIMITER ;
```

What happens here?

Stored Procedures: Parameters

- Stored procedure can have parameters (like methods in programming languages)
- Example: Find employees with salary greater than a certain value sal

```
DELIMITER //
CREATE PROCEDURE GetEmpWithSal( sal FLOAT )
 BEGIN
 SELECT fname, lname, salary
 FROM   employee
 WHERE  salary > sal;
 END //
DELIMITER ;
```

Stored Procedure: Parameter Modes

3 modes (ways) to pass in a parameter

- **IN**: parameter passed by value so the original copy of the parameter value cannot be modified
  (this is the default mode)
- **OUT**: parameter is passed by reference and can be modified by the procedure
  - Assumes OUT parameter is not initialized
- **INOUT**: parameter passed by reference and can be modified but the assumption is that it has been initialized

Syntax:
MODE <varname> DataType

Example: Parameter OUT

```
DELIMITER //

CREATE PROCEDURE OutParam1( IN  x INT,
                            OUT o FLOAT )
 BEGIN
 SELECT max(salary)  INTO  o
 FROM   employee
 WHERE  dno = x;
 END //

DELIMITER ;
```

Stored Procedures: IF Statement

- **IF** statement has the same meaning as ordinary programming language
- IF syntax:

```
IF <condition> THEN
   <command>
END IF;
```

- IF-ELSE statement

```
IF <condition> THEN
    <command1>
ELSE
    <command2>
END IF;
```

Stored Procedure: IF Statement (2)

- Cascaded IF-ELSE statement syntax:

```
IF <condition1> THEN
   <command1>
ELSEIF <condition2> THEN
   <command2>
…
ELSE
   <commandN>
END IF;
```

Example: IF Statement

```
DELIMITER //
CREATE PROCEDURE GetEmpSalLevel( IN essn CHAR(9),
                                 OUT salLevel VARCHAR(9) )
  BEGIN
    DECLARE empSalary DECIMAL(7,2);
    SELECT salary INTO empSalary
    FROM employee
    WHERE ssn = essn;
    IF empSalary < 30000 THEN
      SET salLevel = "Junior";
    ELSEIF (empSalary >= 30000 AND empSalary <= 40000) THEN
      SET salLevel = "Associate";
    ELSE
      SET salLevel = "Executive";
    END IF;
  END //
DELIMITER ;
```

Stored Procedures: CASE Statement

- **CASE** statement is an alternative conditional statement
- Makes code more readable and efficient
- Syntax:

```
CASE <case expression>
    WHEN <expression1> THEN <command1>
    WHEN <expression2> THEN <command2>
    …
    ELSE <commandN>
END CASE;
```

Example: CASE Statement

```
DELIMITER //
CREATE PROCEDURE GetEmpBonus( IN essn CHAR(9),
                              OUT bonus DECIMAL(7,2))
  BEGIN
    DECLARE empDept INT;
    SELECT dno INTO empDept
    FROM employee
    WHERE ssn = essn;
    CASE empDept
      WHEN 1 THEN
         SET bonus = 10000;
      WHEN 4 THEN
         SET bonus = 5000;
      ELSE
         SET bonus = 0;
    END CASE;
  END //
DELIMITER ;
```

Stored Procedure: LOOP statement

3 forms of loops in stored procedures

- WHILE syntax:

```
WHILE <condition> DO
    <commands>
END WHILE;
```

- Repeat until syntax:

```
REPEAT
   <commands>
UNTIL <condition>
END REPEAT;
```

Stored Procedure: LOOP statement (2)

```
<LoopLabel>:
LOOP                                  infinite loop
   <commands>
  IF <condition1> THEN
     LEAVE <LoopLabel>;               works like a break
  IF <condition2> THEN
     ITERATE <LoopLabel>;             works like continue
END LOOP;
```

Example: Loop-Leave Statement

```
DELIMITER //
CREATE PROCEDURE LOOPLoopProc()
   BEGIN
   DECLARE x  INT ;
   SET  x = 0;
   L:  LOOP
      SET  x = x + 1;
      IF  (x >= 5) THEN
       LEAVE L;
      END IF;
      IF  (x mod 2 = 0) THEN
        ITERATE L;
      END IF;
      SELECT x;
      END LOOP;
   END //
DELIMITER ;
```

Cursors: Processing Data

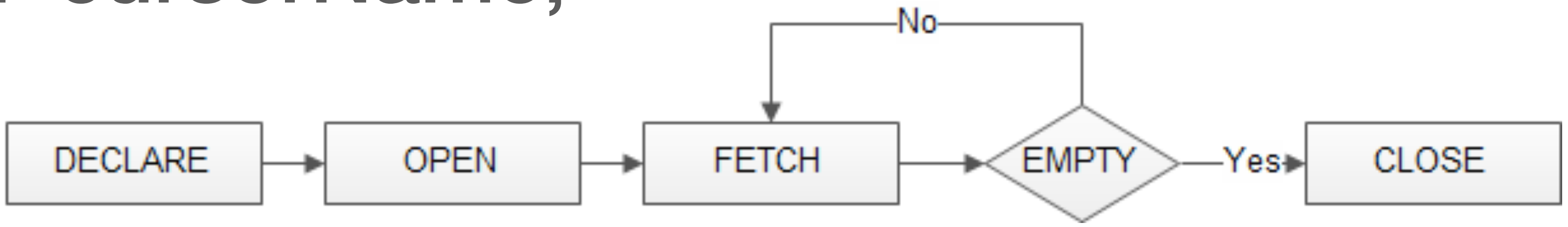
- Programming construct in stored procedures that allow you to iterate through a result set returned by a SQL query
- Read-only data structure (not updatable)
- Non-scrollable: can only be traversed in one direction and cannot skip rows
- Asensitive: server may or may not make a copy of its result table

Working with Cursors

- Declare a cursor using **DECLARE** statement:
  DECLARE <cursor_name> CURSOR FOR <select statement>;
  - Cursor declaration must follow all variable declarations
  - Cursor must always be associated with a SELECT statement
- Declare a handler for the NOT FOUND error condition so that you can exit when the result has been read completely
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

Working with Cursors (2)

- Open the cursor using **OPEN** statement
  OPEN <cursor_name>;
    - Executes the query associated with the cursor
- Use **FETCH** to retrieve the next tuple from cursor data
  FETCH <cursor_name> INTO list-of-variables;
- Close the cursor using **CLOSE** statement
  CLOSE cursorName;



Example: Cursor

```
DELIMITER //
CREATE PROCEDURE cursor1()
BEGIN
DECLARE finished  INTEGER   DEFAULT 0;
DECLARE fname1    CHAR(20)  DEFAULT "";
DECLARE lname1    CHAR(20)  DEFAULT "";
DECLARE nameList  CHAR(100) DEFAULT "";
-- 1. Declare cursor for employee
DECLARE emp_cursor CURSOR FOR SELECT fname, lname FROM employee WHERE salary > 40000;
-- 2. Declare NOT FOUND handler
DECLARE CONTINUE HANDLER  FOR NOT FOUND SET finished = 1;
-- 3. Open the cursor
OPEN emp_cursor;
L: LOOP
     -- 4. Fetch next tuple
  FETCH emp_cursor INTO fname1, lname1;
     -- Handler will set finished = 1 if cursor is empty
  IF finished = 1 THEN
     LEAVE L;
  END IF;
   -- build emp list
   SET nameList = CONCAT( nameList, fname1, ' ', lname1, ';' );
   END LOOP ;
-- 5. Close cursor when done
CLOSE emp_cursor;
SELECT nameList ;
END //
DELIMITER ;
```

Stored Function

- User-defined functions
  - Special stored program that returns a single value (similar to aggregate functions)
  - Meant to encapsulate common formulas or business rules that are reusable
- Syntax:

```
CREATE FUNCTION <function_name>(parameter)
   RETURNS datatype
   [NOT] DETERMINISTIC
<statements>;
```

Example: Stored Function

```
DELIMITER //
CREATE FUNCTION
employeeRaise(salary DECIMAL(7,2))
  RETURNS DECIMAL(7,2) DETERMINISTIC
  BEGIN
  RETURN (1.1 * salary);
  END //

DELIMITER ;
```

MySQL Stored Procedures: Recap

- Session Variables
- Stored Procedures
  - Local variables
  - Parameters
  - IF / CASE / Loop
- Stored Function