MySQL

Principles of Database Design

Ehsan Edalat

Parham Alvani

Outline

- View
- Stored Procedures
- Functions

View

- Create
- Update
- Drop

Create View


```
create view
    passed_std as
    select std_num,
    grade from student
    where grade >=10;
```

Update view


```
create or replace view
    passed_std as select
    std_num, grade, age
    from student where
    grade >=10;
```

Drop view


```
drop view passed_std;
```

Stored Procedures

- Putting database-intensive operations into stored procedures lets you define an API for your database application.
- You can reuse this API across multiple applications and multiple programming languages.

Stored Procedures

- Picking a Delimiter
- Creating a Stored Procedure
- Calling a Stored Procedure
- Delete a Stored Procedure

Picking a Delimiter

- The delimiter is the character or string of characters that you'll use to tell the mySQL client that you've finished typing in an SQL statement.
- For ages, the delimiter has always been a semicolon.
- That, however, causes problems, because, in a stored procedure, one can have many statements, and each must end with a semicolon.

Stored Procedures


```
DELIMITER //
CREATE PROCEDURE p2 ()
COMMENT 'A procedure'
BEGIN
    SELECT 'Hello World !';
END //
```

Stored Procedures


```
DELIMITER //
CREATE PROCEDURE simple (OUT param1 INT)
BEGIN
     SELECT COUNT (*) INTO param1 FROM t;
END //
```

Parameter & Variables

- Parameters
  - Syntax:

    (…, Mode Name Type, …)
  - Modes
    - IN -> call by value
    - OUT -> return value ☺
    - INOUT -> call by reference ☺ ☺
- Variables
  - Declaration Syntax:

    DECLARE Name Type DEFAULT Default value;
  - **Set Syntax:**

    SET Name = Value;

Conditions: IF


```
IF expression
     THEN statements;
ELSIF elsif_expression
     THEN   elsif_statements; ...
ELSE else_statements;
END IF;
```

Conditions: IF Example


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

Conditions: Case


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

Conditions: Case Example


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

Loops: While


```
WHILE expression DO
     statements
END WHILE;
```

Loops: While Example


```
WHILE x  <= 5 DO
     SET  str = CONCAT(str, x, ',');
     SET  x = x + 1;
END WHILE;
```

Calling Procedures

- To call a procedure:
  - you only need to enter the word CALL
  - followed by the name of the procedure
  - and then the parentheses, including all the parameters between them (variables or values).

Calling Procedures


```
CALL stored_procedure_name (param1, param2, ....)

CALL procedure1(10 , 'string parameter' , @parameter_var);
```

Deleting Procedures


```
DROP PROCEDURE IF EXISTS p2;
```

Functions


```
CREATE FUNCTION
      function_name(param1,param2,…)
            RETURNS datatype
            [NOT] DETERMINISTIC
statements
```

Functions


```
CREATE FUNCTION hello (s CHAR(20))
RETURNS CHAR(50) DETERMINISTIC
     RETURN CONCAT('Hello, ',s,'!');
```

Deterministic or Non Deterministic

- Deterministic functions always return the same result any time they are called with a specific set of input values.
- Nondeterministic functions may return different results each time they are called with a specific set of input values.

Function vs. Stored Procedure
Based on stackoverflow answer

- Functions are computed values and cannot perform permanent environmental changes to SQL Server (i.e. no INSERT or UPDATE statements allowed).
- A Function can be used inline in SQL Statements if it returns a scalar value or can be joined upon if it returns a result set.

Function vs. Stored Procedure

- Parameter types and function return types can be declared to use any valid data type.
- The routine_body consists of a valid SQL routine statement. This can be a simple statement such as SELECT or INSERT, or a compound statement written using BEGIN and END.
- MySQL permits routines to contain DDL statements, such as CREATE and DROP. MySQL also permits stored procedures (but not stored functions) to contain SQL transaction statements such as COMMIT.

Function vs. Stored Procedure

- Statements that return a result set can be used within a stored procedure but not within a stored function. This prohibition includes SELECT statements that do not have an INTO var_list clause and other statements such as SHOW, EXPLAIN, and CHECK TABLE.