File Edit Selection View Go Run Terminal Help
Settings - project - code-server
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
mysql

```
|        7 | Cl1       | P4         |        22 | 1200.00 | 2020-09-10 |
|        8 | Cl3       | P1         |        15 |  150.00 | 2020-09-10 |
|        9 | Cl1       | P1         |        10 |  500.00 | 2020-09-12 |
|       10 | Cl2       | P2         |         5 |  100.00 | 2020-09-13 |
+---------+----------+-----------+----------+--------+-----------+
10 rows in set (0.00 sec)

mysql> DELIMITER //
mysql> CREATE FUNCTION GetTotalCost(Cost DECIMAL(5,2)) RETURNS DECIMAL(5,2) DETERMINISTIC BEGIN IF
(Cost >= 100 AND Cost < 500) THEN SET Cost = Cost - (Cost * 0.1); ELSEIF (Cost >= 500) THEN SET Cos
t = Cost - (Cost * 0.2); END IF; RETURN (Cost); END//
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT GetTotalCost(500);
+-------------------+
| GetTotalCost(500) |
+-------------------+
|            400.00 |
+-------------------+
1 row in set (0.00 sec)

mysql> DROP FUNCTION GetTotalCost;
```

Question

When creating complex stored procedures, you must change the delimiter from a semi-colon to another delimiter sign so that MySQL can compile your code in a BEGIN-END block as one compound statement.

[✓ True]

False

Correct

Correct! MySQL requires you to change the delimiter sign so that it can compile your code.

Skip
Continue

File Edit Selection View Go Run Terminal Help
Settings - project - code-server
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
mysql
2:17 / 2:42

```
+-----------+--------------------------+-------+---------------+
| ProductID | ProductName              | Price | NumberOfItems |
+-----------+--------------------------+-------+---------------+
| P1        |  Artificial grass bags   | 50.00 |           100 |
| P2        | Wood panels              | 20.00 |           250 |
| P3        | Patio slates             | 40.00 |            60 |
| P4        | Sycamore trees           | 10.00 |            50 |
| P5        | Trees and Shrubs         | 50.00 |            75 |
| P6        | Water fountain           | 80.00 |            15 |
+-----------+--------------------------+-------+---------------+
6 rows in set (0.00 sec)

mysql> DELIMITER //
mysql> CREATE PROCEDURE GetProductSummary(OUT NumberOfLowPriceProducts INT, OUT NumberOfHighPricePr
oducts INT) BEGIN SELECT COUNT(ProductID) INTO NumberOfLowPriceProducts FROM Products WHERE Price <
50; SELECT COUNT(ProductID) INTO NumberOfHighPriceProducts FROM Products WHERE Price >= 50; END //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL GetProductSummary(@TotalNumberOfLowPriceProducts, @TotalNumberOfHighPriceProducts);
Query OK, 1 row affected (0.00 sec)

mysql>
```

coursera-machin | Face Detection u | Image Threshold | (16) SQL For | (16) Project 1: En | sklearn.model_sc | Differences | (16) Bollywo

coursera.org/learn/advanced-mysql-topics/supplement/LAuLa/differences-between-functions-and-sto...

coursera | Meta | Search in course | Search | Aniket Kolte

Advanced MySQL Topics > Week 1 > Differences between functions and stored procedures | Previous | Next

Video: Variables and parameters
6 min

Video: Developing user-defined functions
4 min

Video: Create complex stored procedures
2 min

Reading: Differences between functions and stored procedures
10 min

Practice Quiz: Knowledge check: MySQL functions and stored procedures
5 questions

Lab: Developing functions in MySQL
1h

the body of the procedure.

In this context, key differences between functions and procedures are as follows:

- A function returns a single value, whereas a procedure may return a single value, multiple values or no value.
- Typically, functions encapsulate common formulas or generic business rules that are reusable among SQL statements and stored procedures. Procedures, on the other hand, are used mainly to process, manipulate and modify data in the database.
- Functions only accept input parameters, while stored procedures can accept IN, OUT and INOUT parameters.
- Functions can be invoked from anywhere, including SELECT statements and stored procedures. Stored procedures are invoked using the CALL statement only.
- A stored function is created using the CREATE FUNCTION statement. A stored procedure is created using the CREATE PROCEDURE statement.
- To build a function, you should specify if it is a DETERMINISTIC function or not. This means that you need to decide if the function always returns the same result for the same input parameters. If you don't use DETERMINISTIC, then MySQL uses the NOT DETERMINISTIC option by default.
- To build functions you must specify the data type of the return value in the RETURNS statement. This can be any valid MySQL data type. However, there's no need to do this with stored procedures.

coursera.org/learn/advanced-mysql-topics/supplement/LAuLa/differences-between-functions-and-sto...

coursera | Meta

Search in course | Search

Aniket Kolte

Advanced MySQL Topics > Week 1 > Differences between functions and stored procedures

< Previous | Next >

Video: Variables and parameters
6 min

Video: Developing user-defined functions
4 min

Video: Create complex stored procedures
2 min

Reading: Differences between functions and stored procedures
10 min

Practice Quiz: Knowledge check: MySQL functions and stored procedures
5 questions

Lab: Developing functions in MySQL
1h

The following table provides a summary of the key differences between stored procedures and stored functions.

	Functions	**Procedures**
1	Created using CREATE FUNCTION command	Created using the CREATE PROCEDURE command
2	Invoked using the SELECT statement	Invoked using the CALL statement
3	Must return a single value	Outputs values via the OUT parameter
4	Takes IN parameters only	Takes IN, OUT and INOUT parameters
5	Typically encapsulates common formulas or generic business rules	Typically used to process, manipulate and modify data in the database
6	Must specify the data type of the return value	User must specify the OUT paameter type

Conclusion

Functions and procedures are used to encapsulate code that can be executed to implement repetitive tasks such as equations, formulas or business rules.

17°C Clear | ENG IN | 23:22 03-11-2022

Lucky Shrub and MySQL trigger types

```
CREATE TRIGGER AfterDeleteOrder
AFTER DELETE
ON Orders FOR EACH ROW
INSERT INTO Audits
VALUES(‘AFTER’, CONCAT('Order',OLD.OrderID, 'was deleted at’, CURRENT_TIME(),
'on’, CURRENT_DATE()), ‘DELETE’);
```

File Edit Selection View Go Terminal Help
Settings - project - code-server
PROBLEMS OUTPUT TERMINAL
mysql
2:33 / 3:12

```
|       18 | Cl1      | P4       |        22 | 1200.00 | 2022-09-10 |
|       19 | Cl3      | P1       |        15 |  150.00 | 2022-09-10 |
|       20 | Cl1      | P1       |        10 |  500.00 | 2022-09-12 |
|       21 | Cl2      | P2       |         5 |  100.00 | 2022-09-13 |
|       22 | Cl2      | P1       |        10 |  500.00 | 2021-09-01 |
|       23 | Cl2      | P2       |         5 |  100.00 | 2021-09-05 |
|       24 | Cl3      | P3       |        20 |  800.00 | 2021-09-03 |
|       25 | Cl4      | P4       |        15 |  150.00 | 2021-09-07 |
|       26 | Cl1      | P3       |        10 |  450.00 | 2021-09-08 |
|       27 | Cl2      | P1       |        20 | 1000.00 | 2022-09-01 |
|       28 | Cl2      | P2       |        10 |  200.00 | 2022-09-05 |
|       29 | Cl3      | P3       |        20 |  800.00 | 2021-09-03 |
|       30 | Cl1      | P1       |        10 |  500.00 | 2022-09-01 |
+----------+----------+----------+-----------+---------+------------+
30 rows in set (0.02 sec)

mysql> DELIMITER //
mysql> CREATE TRIGGER OrderQtyCheck BEFORE INSERT ON Orders FOR EACH ROW BEGIN IF NEW.Quantity < 0
THEN SET NEW.Quantity = 0; END IF; END //
Query OK, 0 rows affected (0.03 sec)

mysql> DELIMITER ;
mysql>
```

Lucky Shrub and one-time scheduled events syntax

```
CREATE EVENT GenerateRevenueReport
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 12 HOUR
DO
BEGIN
  INSERT INTO ReportData (OrderID,ClientID, ProductID,Quantity, Cost, Date)
  SELECT *
  FROM Orders
  WHERE Date
  BETWEEN '2022-08-01' AND '2022-08-31';
END
```

2:40 / 5:50

Lucky Shrub and recurring scheduled event syntax

```
CREATE EVENT DailyRestock
ON SCHEDULE
EVERY 1 DAY
DO
BEGIN
  IF Products.NumberOfItems < 50 THEN
  UPDATE Products SET NumberOfItems = 50;
  END IF;
END
```

File
Edit
Selection
View
Go
Run
Terminal
Help
Settings - project - code-server
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
mysql
4:22 / 4:36

```
 | Using where |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+---------
-+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> CREATE INDEX IdxFullName ON Clients(FullName);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN SELECT ContactNumber FROM Clients WHERE FullName='Jane Delgado';
+----+-------------+---------+------------+------+---------------+-------------+---------+-------+------+-
---------+-------+
| id | select_type | table   | partitions | type | possible_keys | key         | key_len | ref   | rows |
filtered | Extra |
+----+-------------+---------+------------+------+---------------+-------------+---------+-------+------+-
---------+-------+
|  1 | SIMPLE      | Clients | NULL       | ref  | IdxFullName   | IdxFullName | 803     | const |    1 |
  100.00 | NULL  |
+----+-------------+---------+------------+------+---------------+-------------+---------+-------+------+-
---------+-------+
1 row in set, 1 warning (0.00 sec)

mysql> 
```

Lucky Shrub and MySQL transactions

```
START TRANSACTION
INSERT INTO Orders (OrderID, ClientID, ProductID , Quantity, Cost, Date)
VALUES (22, "Cl1", "P1", 10, 500, "2022-09-01" );
UPDATE Products SET NumberOfItems = (NumberOfItems - 10) WHERE ProductID =
"P1";
SELECT Orders.OrderID, Orders.Quantity, Products.ProductID,
Products.NumberOfItems FROM Orders INNER JOIN Products ON (Orders.ProductID =
Products.ProductID) WHERE Orders.OrderID = 22;
```

Lucky Shrub and Common Table Expressions

```
WITH
Average_Sales_2020 AS (SELECT CONCAT(AVG(Cost), "in 2020") AS "Average Sale“ FROM
Orders WHERE YEAR(Date) = 2020),
Average_Sales_2021 AS (SELECT CONCAT(AVG(Cost), "in 2021") FROM Orders WHERE
YEAR(Date) = 2021),
Average_Sales_2022 AS (SELECT CONCAT(AVG(Cost), "in 2022") FROM Orders WHERE
YEAR(Date) = 2022)
SELECT * FROM Average_Sales_2020
UNION
SELECT * FROM Average_Sales_2021
UNION
SELECT * FROM Average_Sales_2022;
```

2:18 / 4:07

FULL OUTER JOIN

A JOIN used to return all records from two tables, including those that don't have a match.

FULL OUTER JOIN with UNION ALL operator

```
SELECT columns
FROM table1
LEFT JOIN table2
ON table1.CommonColumn = table2.CommonColumn
UNION ALL
SELECT columns
FROM table1
RIGHT JOIN table2
ON table1.CommonColumn = table CommonColumn.CommonColumn
```

2:19 / 4:26

FULL OUTER JOIN with UNION operator

```
SELECT columns
FROM table1
LEFT JOIN table2
ON table1.CommonColumn = table2.CommonColumn
UNION
SELECT columns
FROM table1
RIGHT JOIN table2
ON table1.CommonColumn = table CommonColumn.CommonColumn
```

File Edit Selection View Go Run Terminal Help
Settings - project - code-server
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
mysql

```
+----------+-----------------+---------------+----------+
6 rows in set (0.00 sec)

mysql> SELECT * FROM Products;
+-----------+-----------------------+----------+-----------+---------------+
| ProductID | ProductName           | BuyPrice | SellPrice | NumberOfItems |
+-----------+-----------------------+----------+-----------+---------------+
| P1        | Artificial grass bags |    40.00 |     50.00 |           100 |
| P2        | Wood panels           |    15.00 |     20.00 |           250 |
| P3        | Patio slates          |    35.00 |     40.00 |            60 |
| P4        | Sycamore trees        |     7.00 |     10.00 |            50 |
| P5        | Trees and Shrubs      |    35.00 |     50.00 |            75 |
| P6        | Water fountain        |    65.00 |     80.00 |            15 |
+-----------+-----------------------+----------+-----------+---------------+
6 rows in set (0.00 sec)

mysql> SELECT Clients.ClientID, Clients.ContactNumber,
    -> Orders.OrderID, Orders.Quantity, Orders.Date,
    -> Products.NumberOfItems AS 'Items in stock' FROM
    -> Clients INNER JOIN Orders INNER JOIN Products
    -> ON(Clients.ClientID = Orders.ClientID AND Orders.ProductID = Products.ProductID)
    -> WHERE(Orders.Quantity >= 10 AND Date > '2020-09-05' AND Products.NumberOfItems >= 50);
```

File Edit Selection View Go Run Terminal Help
Settings - project - code-server
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
mysql
2:45 / 2:51

```
|       28 | Cl2      | P2         |       10 |  200.00 | 2022-09-05 |
|       29 | Cl3      | P3         |       20 |  800.00 | 2021-09-03 |
|       30 | Cl1      | P1         |       10 |  500.00 | 2022-09-01 |
+---------+----------+-----------+----------+--------+-----------+
30 rows in set (0.00 sec)

mysql> PREPARE GetOrderStatement FROM 'SELECT ClientID, ProductID, Quantity, Cost FROM Orders WHERE
 OrderID = ?';
Query OK, 0 rows affected (0.00 sec)
Statement prepared

mysql> SET @order_id = 10;
Query OK, 0 rows affected (0.00 sec)

mysql> EXECUTE GetOrderStatement USING @order_id;
+----------+-----------+----------+--------+
| ClientID | ProductID | Quantity | Cost   |
+----------+-----------+----------+--------+
| Cl2      | P2        |        5 | 100.00 |
+----------+-----------+----------+--------+
1 row in set (0.00 sec)

mysql>
```

coursera-mac
Face Detectio
Image Thresh
SQL For Da
Project 1: End
sklearn.model
Additional res
MySQL Pr
Displaying Im
mysqltutorial.org/mysql-prepared-statement.aspx
LIKE
LIMIT
IS NULL
Table & Column Aliases
Joins
INNER JOIN
LEFT JOIN
RIGHT JOIN
Self Join
CROSS JOIN
GROUP BY
HAVING
ROLLUP
Subquery
Derived Tables
18°C
Clear
ENG
IN
21:43
06-11-2022

The prepared statement takes advantage of client/server binary protocol. It passes the query that contains placeholders (`?`) to the MySQL Server as the following example:

```
SELECT *
FROM products
WHERE productCode = ?;
```

When MySQL executes this query with different `productcode` values, it does not have to fully parse the query. As a result, this helps MySQL execute the query faster, especially when MySQL executes the same query multiple times.

Since the prepared statement uses placeholders (`?`), this helps avoid many variants of SQL injection hence make your application more secure.

MySQL prepared statement usage

In order to use MySQL prepared statement, you use three following statements:

- `PREPARE` – prepare a statement for execution.
- `EXECUTE` – execute a prepared statement prepared by the `PREPARE` statement.
- `DEALLOCATE PREPARE` – release a prepared statement.

```sql
CREATE TRIGGER before_employee_update
    BEFORE UPDATE ON employees
    FOR EACH ROW
 INSERT INTO employees_audit
 SET action = 'update',
     employeeNumber = OLD.employeeNumber,
     lastname = OLD.lastname,
     changedat = NOW();
```