Indian Institute of Information Technology Sri City

Database Management Systems LAB-10

Date: 09/11/2021

TOPIC: Revision of Stored procedures, Error Handling, Triggers and Cursors

Instructors: Dr. Odelu Vanga, Dr. Rakesh, Dr. Annushree

STORED PROCEDURES AND FUNCTIONS

EXAMPLE 1:

Display details of all books

```
delimiter //
create procedure display_book()
begin
select *from book;
end //
```

EXAMPLE 2:

Update price of a book taking ISBN of the book and its new price as input

```
delimiter //
create procedure update_price (IN temp_ISBN varchar(10), IN new_price integer)
begin
update book set price=new_price where ISBN=temp_ISBN;
end //
```

EXAMPLE 3:

Display the highest price among all the books with an output parameter

```
delimiter //
create procedure disp_max(OUT highestprice integer)
begin
select max(price) into highestprice from book;
end //
```

EXAMPLE 4:

Accept gender as input and display no.of authors having the given gender

```
delimiter //
create procedure disp_gender(INOUT mfgender integer, IN emp_gender varchar(6))
begin
select count(gender) into mfgender from author where gender emp_gender;
end //
create function my_fun(emp_gender varchar(6))
returns int
deterministic
```

```
begin
declare r int;
select count(gender) into r from author where gender = emp_gender;
return (r);
end//
```

TRIGGERS:

Interesting Key points about Triggers:

- Triggers are similar to stored procedures but differ in the way that they are invoked.
- Trigger can only be associated with a table and defined to fire when an INSERT, DELETE or UPDATE statement is performed on the table.
- MySQL does not permit two triggers with the same trigger timing (BEFORE or AFTER) and trigger event or statement (INSERT, DELETE, or UPDATE) to be defined on a table.
- For example, you cannot define two BEFORE INSERT or two AFTER UPDATE triggers for a table.
- All triggers defined on MySQL are row triggers, which mean that the action defined for the triggers is executed for each row affected by the triggering statement.

STORED PROCEDURE VS TRIGGER:

A stored procedure is a user defined piece of code written in the local version of PL/SQL, which may return a value (making it a function) that is invoked by calling it explicitly.

A trigger is a stored procedure that runs automatically when various events happen (eg update, insert, delete).

EXAMPLE:

Let's create a trigger named **updateItemPrice**. This particular trigger is activated whenever the **items** table is updated. When this event occurs, the trigger checks each row to see if the product cost (c**ost**) value is being changed. If it is, then the trigger automatically sets the item's new price (**price**) to 1.40 times the item's new cost (in other words, a 40% markup).

To create this trigger, run the following MySQL statements:

```
DELIMITER $$
CREATE TRIGGER `updateItemPrice`
BEFORE UPDATE ON `items`
FOR EACH ROW
BEGIN
```

```
  IF NEW.cost <> OLD.cost
     THEN
       SET NEW.price = NEW.cost * 1.40;
  END IF ;
END$$
DELIMITER ;
```

Using the trigger

The **updateItemPrice** trigger is now ready to be invoked automatically whenever a row in the **products** table is updated. For example, run the following SQL statement to change the cost of the Basic Widget:

```
UPDATE items SET cost = 7.00 WHERE id = 1;
```

When you run this SQL statement, the trigger activates as well, and automatically updates the Basic Widget's price in proportion to the new cost. To verify this, you can run the following SQL statement:

```
SELECT * FROM items;
```

ERROR HANDLING:

EXAMPLE:

```
drop procedure emp_details;
DELIMITER //
CREATE PROCEDURE emp_details
(
     InputID INTEGER
    ,InputName VARCHAR(50)
    ,InputDept VARCHAR(50)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    SELECT 'Error occured';
    INSERT INTO employees VALUES(InputID, InputName, InputDept);
    SELECT *FROM employees;
END
// DELIMITER ;
```

Now, call the above SP two times with same ID.

The first time, it will execute successfully, but the second time it will throw a custom error message.

```
call emp_details(200,'Johny','HR');
call emp_details(200,'Johny','HR');
```

MySQL Cursor

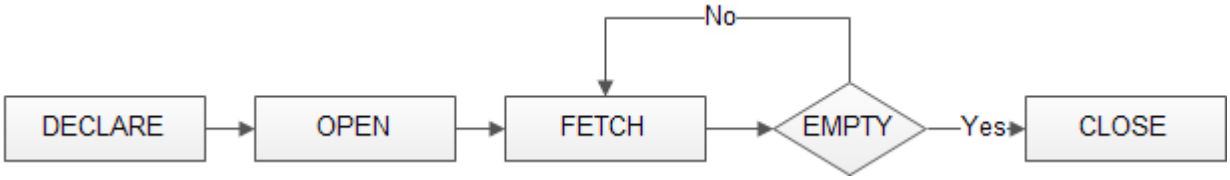
To handle a result set inside a stored procedure, you use a cursor. A cursor allows you to iterate a set of rows returned by a query and process each row individually.

MySQL cursor is

- read-only
- non-scrollable and
- Asensitive.

Working with MySQL cursor

- ✓ `DECLARE cursor_name CURSOR FOR SELECT_statement;`
- ✓ `OPEN cursor_name;`
- ✓ `FETCH cursor_name INTO variables list;`
- ✓ `CLOSE cursor_name;`



Example:

Cursor to iterate emp name and their place:

```
DROP procedure if exists emp_curs;
DELIMITER $$
CREATE PROCEDURE emp_curs()
BEGIN
      DECLARE finished INTEGER DEFAULT 0;
      DECLARE ename varchar(100);
    DECLARE eplace varchar(100);

      -- declare cursor for employee name and place
      DEClARE curname CURSOR FOR SELECT emp_name, place FROM employee;
      -- declare NOT FOUND handler
      DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;
      OPEN curname;
```

```
      getname: LOOP
            FETCH curname INTO ename, eplace;
            IF finished = 1 THEN
                  LEAVE getname;
            END IF;
            -- build employee names
            SELECT ename,eplace;
      END LOOP getname;
      CLOSE curname;

END$$
DELIMITER ;
```

Example 2:

Let's say we sell products of some types. We want to count how many products of each type are exists.

```
DELIMITER //
DROP PROCEDURE IF EXISTS product_count;
CREATE PROCEDURE product_count()
  BEGIN
    DECLARE p_type VARCHAR(255);
    DECLARE p_count INT;
    DECLARE done INT DEFAULT 0;
    DECLARE product_curs CURSOR FOR
      SELECT type,COUNT(*)FROM product GROUP BY type;
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;
    TRUNCATE product_type;
    OPEN product_curs;
    REPEAT
      FETCH product_curs
      INTO p_type, p_count;
      IF NOT done
      THEN
        INSERT INTO product_type_count
        SET
          type  = p_type,
          count = p_count;
      END IF;
    UNTIL done
    END REPEAT;
    CLOSE product_curs;
  END //
```

```
DELIMITER ;
```

When you may call procedure with:

```
CALL product_count();
```

Result would be in product_type_count table.

References:

Course material is inspired from the following websites and some examples are taken from geeksforgeeks, w3schols only for student understanding perspective.

1. https://www.mysqltutorial.org/mysql-error-handling-in-stored-procedures/
2. https://www.mysqltutorial.org/mysql-cursor/

Some part of the exercises are inspired from University of GothenBurg

3. http://www.cse.chalmers.se/edu/year/2018/course/TDA357/VT2018/tutorials
4. https://www.mysqltutorial.org
5. https://www.brainbell.com/tutorials/MySQL/Working_With_Cursors.htm