MySQL for Developers

SQL-4501 Release 2.2

D61830GC10
Edition 1.0

Day 3

- Stored Procedures / Functions
- Triggers
- Events

Stored Routines

What is a Stored Routine?

- Set of SQL statements that can be stored in server
- Types
  - Stored procedures
    - A procedure is invoked using a call statement, and can only pass back values using output variables
  - Stored functions
    - A function can be called from inside a statement and can return a scalar value

Creating Procedures

```
drop procedure if exists display_emp_info;
delimiter $
CREATE PROCEDURE display_emp_info(p_id integer)
BEGIN
  Select ename, salary
  from emp
   where id = p_id;
END$
delimiter ;
```

Invoking Procedure

```
Call display_emp_info(1);
```

Creating Function

```
drop function if exists tax;

delimiter $

CREATE FUNCTION tax(p_id integer)

RETURNS int(11)

BEGIN

  RETURN p_id * 0.1 ;

END$

delimiter ;
```

Invoking Function

```
Select Tax (1000);

Select Tax(Salary) from emp;
```

Creating Function

```
drop function if exists thank_you;

delimiter $

CREATE FUNCTION thank_you(p_name char(50))

RETURNS char(100)

BEGIN

RETURN CONCAT('Thank You, ',p_name,'!');

END$

delimiter ;
```

Invoking Function

```
Select thank_you(name) from emp;
```

Compound statements

```
drop procedure if exists multitask;
delimiter $
CREATE procedure multitask()
BEGIN
   select * from emp;
   select * from dept;
   call display_emp_info(1);
   select tax(salary) from emp;
   Select thank_you(name) from emp;
END$
delimiter ;
```

Declaring Variables

```
DELIMITER //

CREATE FUNCTION add_tax (total_charge FLOAT(9,2))
RETURNS FLOAT(10,2)
BEGIN
  DECLARE tax_rate FLOAT (3,2) DEFAULT 0.07;
  RETURN total_charge + total_charge * tax_rate;

END//

DELIMITER ;
```

Assign Variables (SELECT ... INTO / SET)

```
CREATE procedure display_dept_name(p_id integer)
BEGIN
  Declare v_dno integer;
  Declare v_name varchar(50);
  SET v_name = (select ename
                        from emp
                      where id = p_id);
  select deptno
  into v_dno
  from emp
  where id = p_id;
  /* print*/
  select thank_you(v_name);
  select dname from dept where deptno = v_dno;
END$
delimiter ;
```

Examine Stored Routines

- `SHOW CREATE PROCEDURE` / `FUNCTION`
  - MySQL specific
  - Returns exact code string
- `SHOW PROCEDURE` / `FUNCTION STATUS`
  - MySQL specific
  - Returns characteristics of routines
- `INFORMATION_SCHEMA.ROUTINES`
  - Standard SQL
  - Returns a combination of the `SHOW` commands

Delete Stored Routines

- **DROP PROCEDURE**

```
DROP PROCEDURE [IF EXISTS] procedure_name;
```

  - Example

```
DROP PROCEDURE proc_1;
```

- **DROP FUNCTION**

```
DROP FUNCTION [IF EXISTS] function_name;
```

  - Example

```
DROP FUNCTION IF EXISTS func_1;
```

Flow Control Statements

- Statements and constructs that control order of operation execution
- Common flow controls
  - Choices
    - **`IF`** and **`CASE`**
  - Loops
    - **`REPEAT`**, **`WHILE`** and **`LOOP`**

IF

- The most basic of all choice flow controls or conditional constructs

```
IF (test condition) THEN
ELSEIF (test condition) THEN
ELSE
END IF
```

CASE

- **CASE** provides a means of developing complex conditional constructs
- **CASE** works on the principle of comparing a given value with specified constants and acting upon the first constant that is matched

```
CASE case_value
  WHEN value THEN

  ELSE

END CASE
```

OR

```
CASE
  WHEN test_condition THEN

  ELSE

END CASE
```

REPEAT

- The **REPEAT** statement repeats the statements between the **REPEAT** and **UNTIL** keywords until the condition after the **UNTIL** keyword becomes **TRUE**
- A **REPEAT** loop always iterates at least once
- Optional Labels
  - Begin
  - End

```
my_label: REPEAT

UNTIL test_condition
END REPEAT my_label;
```

WHILE

- **WHILE** repeats the statements between the **DO** and **END WHILE** keywords as long as the condition appearing after the **WHILE** keyword remains **TRUE**
- A **WHILE** loop may never iterate (if the condition is initially **FALSE**)

```
my_label: WHILE test_condition
DO
END WHILE my_label;
```

LOOP

- The statements between the **LOOP** and **END LOOP** keywords are repeated.
- The loop must be explicitly exited, and usually this is accomplished with a **LEAVE** statement.
- A valid label must appear after the **LEAVE** keyword.

```
my_label: LOOP

  LEAVE my_label;
END LOOP my_label;
```

Triggers

What Are Triggers?

- Named database objects
- Activated when table data is modified
- Bring a level of power and security to table data
- Trigger scenario using the world database
  - What would you do after changing the Country table code column?
  - Since the code is stored in all three world database tables, it is best to change all 3 at once
  - A trigger can accomplish this task
- Trigger features

Creating Triggers

- Syntax

```
CREATE TRIGGER trigger_name
  { BEFORE | AFTER }
  { INSERT | UPDATE | DELETE }
  ON table_name
  FOR EACH ROW
  triggered_statement
```

```
create table deleted_emp like emp;
```

```
CREATE TRIGGER emp_deletion_log
AFTER DELETE ON emp
FOR EACH ROW
     INSERT INTO Deleted_emp (ID, eName)
     VALUES (OLD.ID, OLD.eName);
```

To test the trigger

```
delete from emp where id = 6;

select * from deleted_emp;
```

Delete Triggers

- **DROP TRIGGER**

```
DROP TRIGGER trigger_name;
DROP TRIGGER schema_name.trigger_name;
```

If you drop a table, the triggers are automatically dropped also.

Events

```
CREATE EVENT delete_changes
ON SCHEDULE EVERY 48 HOUR
DO
  DELETE FROM changes;
```

Make sure that event scheduler variable is ON

```
select @@global.event_scheduler;
set @@global.event_scheduler =1;
```

GUI Tools

phpMyAdmin

MySQL Workbench