MIT-WPU

॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Dr. Vishwanath Karad

MIT WORLD PEACE UNIVERSITY | PUNE

TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

AID20040

Database Management Systems

DEPARTMENT OF COMPUTER ENGINEERING AND TECHNOLOGY

Syllabus

UNIT IV – INTRODUCTION TO PROGRAMMING IN SQL(PL/SQL)

PL/SQL Concepts: Basic PL/SQL Block, PL/SQL Functions and Procedures, Different modes in Procedures, Difference between Functions and Procedures, Cursors, Database Triggers. Exception Handling in PL/SQL .Application of DBMS

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Extensions to SQL (PL/SQL)

What is PL/SQL

PL/SQL:

- Stands for Procedural Language extension to SQL
- Is Oracle Corporation's standard data access language for relational databases
- Seamlessly integrates procedural constructs with SQL

PL/SQL Functions and Procedures

- Procedural Language extension for SQL
- Oracle Proprietary
- 3GL Capabilities
- Integration of SQL
- Portable within Oracle data bases
- Callable from any client

Structure of PL/SQL

- PL/SQL is Block Structured

A block is the basic unit from which all PL/SQL programs are built. A block can be named (functions and procedures) or anonymous

- Sections of block

1- Header Section

2- Declaration Section

3- Executable Section

4- Exception Section

Structure of PL/SQL

HEADER

Type and Name of block

DECLARE

Variables; Constants; Cursors;

BEGIN

PL/SQL and SQL Statements

EXCEPTION

Exception handlers

END;

Stored Function

PL/SQL Functions

- Functions are declared using the following syntax:

```
Create function <function-name> (param_1, …, param_k)
        returns <return_type>
        [not] deterministic          allow optimization if same output
                                     for the same input (use RAND not deterministic )
Begin
   -- execution code
end;
```

Deterministic and Non- deterministic Functions

- A deterministic function always returns the same result for the same input parameters whereas a non-deterministic function returns different results for the same input parameters.
- If you don't use DETERMINISTIC or NOT DETERMINISTIC, MySQL uses the NOT DETERMINISTIC option by default.
- my**rand()** is nondeterministic function. That means we do not know what it will return ahead of time.

```
DELIMITER $$
CREATE FUNCTION myrand()
RETURNS INT
NOT DETERMINISTIC
READS SQL DATA
BEGIN
    RETURN round(rand()*10000, 0);
END$$

DELIMITER ;
Select myrand();
```

PL/SQL Functions – Example 1

- Define a function that, given the name of a department, returns the count of the number of instructors in that department.

```
create function dept_count (dept_name varchar(20))
        returns integer
    begin
        declare d_count  integer;
                select count (* ) into d_count
                from instructor
                where instructor.dept_name = dept_name
        return d_count;
    end
```

Example 1 (Cont)..

- The function dept_count can be used to find the department names and budget of all departments with more than 12 instructors.

```
select dept_name, budget
from department
where dept_count (dept_name ) > 12
```

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Example 2

- A function that returns the level of a customer based on credit limit. We use the IF statement to determine the credit limit.

```
DELIMITER $$

CREATE FUNCTION CustomerLevel(p_creditLimit double) RETURNS VARCHAR(10)
    DETERMINISTIC
BEGIN
    DECLARE lvl varchar(10);

    IF p_creditLimit > 50000 THEN
 SET lvl = 'PLATINUM';
    ELSEIF (p_creditLimit <= 50000 AND p_creditLimit >= 10000) THEN
        SET lvl = 'GOLD';
    ELSEIF p_creditLimit < 10000 THEN
        SET lvl = 'SILVER';
    END IF;

 RETURN (lvl);
END
```

Example 2 (Cont..)

- **Calling function:**
- we can call the CustomerLevel() in a SELECT statement as follows:

```
1  SELECT
2      customerName,
3      CustomerLevel(creditLimit)
4  FROM
5      customers
6  ORDER BY
7      customerName;
```

Output:

	customerName	CustomerLevel(creditLimit)
▶	Alpha Cognac	PLATINUM
	American Souvenirs Inc	SILVER
	Amica Models & Co.	PLATINUM
	ANG Resellers	SILVER
	Anna's Decorations, Ltd	PLATINUM
	Anton Designs, Ltd.	SILVER

Example 3

```
mysql> select * from employee;
+----+-------+----------+--------+-------------+------+
| id | name  | superid  | salary | bdate       | dno  |
+----+-------+----------+--------+-------------+------+
|  1 | john  |        3 | 100000 | 1960-01-01  |    1 |
|  2 | mary  |        3 |  50000 | 1964-12-01  |    3 |
|  3 | bob   |     NULL |  80000 | 1974-02-07  |    3 |
|  4 | tom   |        1 |  50000 | 1970-01-17  |    2 |
|  5 | bill  |     NULL |   NULL | 1985-01-20  |    1 |
+----+-------+----------+--------+-------------+------+
5 rows in set (0.00 sec)

mysql> delimiter |
mysql> create function giveRaise (oldval double, amount double
    -> returns double
    -> deterministic
    -> begin
    ->         declare newval double;
    ->         set newval = oldval * (1 + amount);
    ->         return newval;
    -> end |
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```

Example 3 (cont..)

```
mysql> select name, salary, giveRaise(salary, 0.1) as newsal
    -> from employee;
+------+--------+--------+
| name | salary | newsal |
+------+--------+--------+
| john | 100000 | 110000 |
| mary |  50000 |  55000 |
| bob  |  80000 |  88000 |
| tom  |  50000 |  55000 |
| bill |   NULL |   NULL |
+------+--------+--------+
5 rows in set (0.00 sec)
```

Stored Procedures

Stored Procedures in MySQL

- A stored procedure contains a sequence of SQL commands stored in the database catalog so that it can be invoked later by a program
- Stored procedures are declared using the following syntax:

```
Create Procedure <proc-name>
      (param_spec1, param_spec2, …, param_specn )
begin
   -- execution code
end;
```

where each param_spec is of the form:

[in | out | inout] <param_name> <param_type>

- in mode: allows you to pass values into the procedure,
- out mode: allows you to pass value back from procedure to the calling program

Example 1 – No parameters

- The GetAllProducts() stored procedure selects all products from the products table.

```
mysql> use classicmodels;
Database changed
mysql> DELIMITER //
mysql> CREATE PROCEDURE GetAllProducts()
    -> BEGIN
    -> SELECT * FROM products;
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql>
```

Example 1 (Cont..)

- **Calling Procedure:**

```
CALL GetAllProducts();
```

- **Output:**

productCode	productName	productLine	productScale
S10_1678	1969 Harley Davidson Ultimate Chopper	Motorcycles	1:10
S10_1949	1952 Alpine Renault 1300	Classic Cars	1:10
S10_2016	1996 Moto Guzzi 1100i	Motorcycles	1:10
S10_4698	2003 Harley-Davidson Eagle Drag Bike	Motorcycles	1:10
S10_4757	1972 Alfa Romeo GTA	Classic Cars	1:10

Example 2 (with IN parameter)

```
mysql> select * from employee;
+----+------+---------+--------+------------+------+
| id | name | superid | salary | bdate      | dno  |
+----+------+---------+--------+------------+------+
|  1 | john |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob  |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom  |       1 |  50000 | 1978-01-17 |    2 |
|  5 | bill |    NULL |   NULL | 1985-01-20 |    1 |
+----+------+---------+--------+------------+------+
```

```
mysql> select * from department;
+---------+-------------+
| dnumber | dname       |
+---------+-------------+
|       1 | Payroll     |
|       2 | TechSupport |
|       3 | Research    |
+---------+-------------+
```

- Suppose we want to keep track of the total salaries of employees working for each department

```
mysql> create table deptsal as
    -> select dnumber, 0 as totalsalary from department;
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |           0 |
|       2 |           0 |
|       3 |           0 |
+---------+-------------+
```

We need to write a procedure to update the salaries in the deptsal table

Example 2 (Cont..)

```
mysql> delimiter //
mysql> create procedure updateSalary (IN param1 int)
    -> begin
    ->     update deptsal
    ->     set totalsalary = (select sum(salary) from employee where dno = param1)
    ->     where dnumber = param1;
    -> end; //
Query OK, 0 rows affected (0.01 sec)
```

1. Define a procedure called updateSalary which takes as input a department number.
2. The body of the procedure is an SQL command to update the totalsalary column of the deptsal table.

Example 2 (Cont..)

Step 3: Call the procedure to update the totalsalary for each department

```
mysql> call updateSalary(1);
Query OK, 0 rows affected (0.00 sec)

mysql> call updateSalary(2);
Query OK, 1 row affected (0.00 sec)

mysql> call updateSalary(3);
Query OK, 1 row affected (0.00 sec)
```

Example 2 (Cont..)

Step 4: Show the updated total salary in the deptsal table

```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

Example 3 (with OUT Parameter)

- The following example shows a simple stored procedure that uses an OUT parameter.
- Within the procedure MySQL MAX() function retrieves maximum salary from job_title of employee table.

```
CREATE PROCEDURE my_proc_OUT (OUT highest_salary INT)
BEGIN
SELECT MAX(salary) INTO highest_salary
    FROM employee;
END$$
```

(Cont..)

- **Procedure Call:**
- mysql> CALL my_proc_OUT(@M)$$
- Query OK, 1 row affected (0.03 sec)
- mysql< SELECT @M$$

- **Output:**

```
+-------+
| @M    |
+-------+
| 40000 |
+-------+
1 row in set (0.00 sec)
```

Example 4 (with INOUT Parameter)

- The following example shows a simple stored procedure that uses an INOUT parameter.
- 'count' is the INOUT parameter, which can store and return values and 'increment' is the IN parameter, which accepts the values from user.

```
mysql> DELIMITER // ;
mysql> Create PROCEDURE counter(INOUT count INT, IN increment INT)
    -> BEGIN
    -> SET count = count + increment;
    -> END //
Query OK, 0 rows affected (0.03 sec)
```

Example 4 (Cont..)

Function Call:

```
mysql> DELIMITER ;
mysql> SET @counter = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL counter(@Counter, 1);
Query OK, 0 rows affected (0.00 sec)

mysql> Select @Counter;
+----------+
| @Counter |
+----------+
| 1        |
+----------+
1 row in set (0.00 sec)
```

Stored Procedures (Cont..)

- Use show procedure status to display the list of stored procedures you have created

```
mysql> show procedure status;
+------+---------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
| Db   | Name          | Type      | Definer | Modified            | Created             | Security_
type | Comment | character_set_client | collation_connection | Database Collation |
+------+---------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
| ptan | updateSalary0 | PROCEDURE | ptan@%  | 2010-03-16 12:21:55 | 2010-03-16 12:21:55 | DEFINER
     |         | latin1               | latin1_swedish_ci    | latin1_swedish_ci  |
+------+---------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
1 row in set (0.02 sec)
```

- Use drop procedure to remove a stored procedure

```
mysql> drop procedure updateSalary;
Query OK, 0 rows affected (0.00 sec)
```

Difference between Functions and Procedures

Parameter	SQL Function	SQL Procedure
Syntax	Defined using the CREATE FUNCTION statement	Defined using the CREATE PROCEDURE statement
Execution	It can be executed from within a SELECT statement or another function	Must be explicitly called using the EXECUTE statement or another procedure
Return Value	Always returns a single value	May or may not return a value
Purpose	Used to perform a single calculation or operation	Used to perform a series of actions
Input Parameter	Can accept input parameters	Can accept input parameters
Output Parameter	Cannot have output parameters	Can have output parameters
Examples	SUM, CONCAT, YEAR	Write procedure that update price of pno by 80% of its old price if qty_in_hand is more than 200 otherwise by 70%.

Language Constructs for Procedures & Functions

- SQL supports constructs that gives it almost all the power of a general-purpose programming language.
  - Warning: most database systems implement their own variant of the standard syntax below.
- Compound statement: **begin … end**,
  - May contain multiple SQL statements between **begin** and **end**.
  - Local variables can be declared within a compound statements

Language Constructs

- **CASE Statement**

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

- **While** and **repeat** statements:

while *boolean expression* **do**
sequence of statements ;
end while

repeat
sequence of statements ;
until *boolean expression*
end repeat

-

Write a procedure to calculate number of employees from jobs table based on the salary provided

```
DELIMITER $$
CREATE PROCEDURE my_proc_CASE (INOUT no_employees INT, IN salary INT)
BEGIN
CASE
WHEN (salary>10000)
THEN (SELECT COUNT(job_id) INTO no_employees
FROM jobs WHERE salary>10000);
WHEN (salary<10000)
THEN (SELECT COUNT(job_id) INTO no_employees
 FROM jobs WHERE salary<10000);
ELSE (SELECT COUNT(job_id) INTO no_employees
FROM jobs WHERE salary=10000);
END CASE;
 END$$
```

Language Constructs (Cont.)

- **Loop, Leave and Iterate statements…**
  - Permits iteration over all results of a query.

```
SET x=0;
loop_label:  LOOP
IF  x > 10 THEN
 LEAVE  loop_label;
END  IF;
SET  x = x + 1;
IF  (x mod 2) THEN
ITERATE  loop_label;
ELSE
        SET  str = CONCAT(str,x,',');
END  IF;
    END LOOP;
```

Language Constructs (Cont.)

- **Conditional statements (if-then-else)**

```
IF expression THEN
  statements;
ELSE
  else-statements;
END IF;
```

Example:

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

Exceptions in PLSQL

- In PL/SQL, an exception is a mechanism for handling runtime errors or unexpected conditions that occur during the execution of a program block.
- **System-Defined (Predefined) Exceptions**
- **User-Defined Exceptions**

Exception Handlers in MySQL

Continue Handler and Exit Handler

A **CONTINUE handler** tells MySQL to continue execution after the exception is handled.

An **EXIT handler** tells MySQL to exit the current block immediately. No further statements are executed. This is used when an error is critical and you don’t want to continue further execution.

Common Error codes in mysql

Error Code	Description
1045	Invalid login credentials.
1054	Column does not exist in the specified table
1062	Duplicate entry for key
1146	Table doesn't exist
1216	Violation of a foreign key constraint during insert or update
1217	Attempt to delete or update a row that is referenced by a foreign key in another table.

Example

- First, create a new table named article_tags.

```
1  CREATE TABLE article_tags(
2      article_id INT,
3      tag_id     INT,
4      PRIMARY KEY(article_id,tag_id)
5  );
```

- The article_tags table stores the relationships between articles and tags. Each article may have many tags and vice versa.
- Next, create a stored procedure that inserts article id and tag id into the article_tags table.

Example (Cont..)

```
1  DELIMITER $$
2
3  CREATE PROCEDURE insert_article_tags(IN article_id INT, IN tag_id INT)
4  BEGIN
5
6   DECLARE CONTINUE HANDLER FOR 1062
7   SELECT CONCAT('duplicate keys (',article_id,',',tag_id,') found') AS msg;
8
9   -- insert a new record into article_tags
10  INSERT INTO article_tags(article_id,tag_id)
11  VALUES(article_id,tag_id);
12
13  -- return tag count for the article
14  SELECT COUNT(*) FROM article_tags;
15 END
```

Example (Cont..)

- Then, Call the procedure

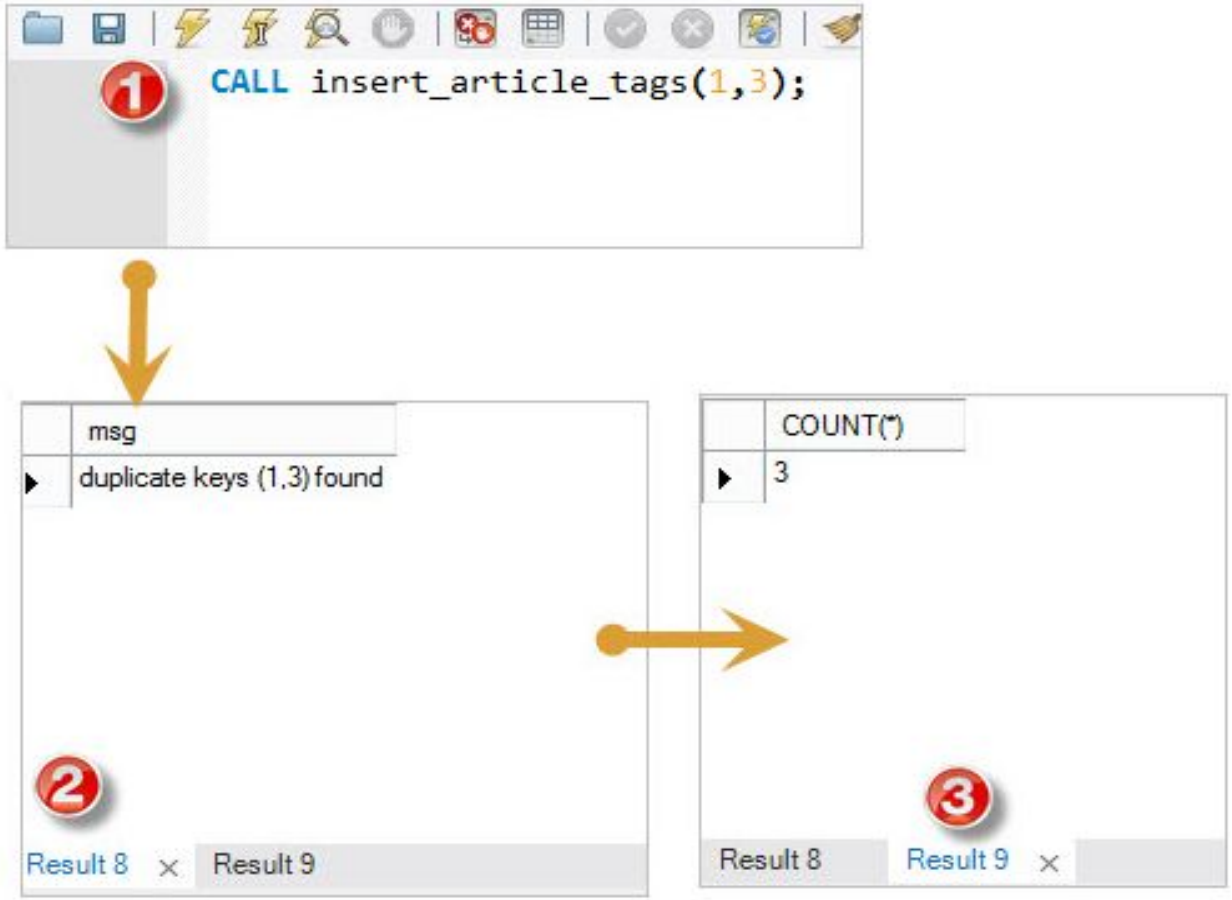
```
1 CALL insert_article_tags(1,1);
2 CALL insert_article_tags(1,2);
3 CALL insert_article_tags(1,3);
```

- After that, try to insert a duplicate key to check if the handler is really invoked.

```
1 CALL insert_article_tags(1,3);
```

- We will get an error message. However, because we declared the handler as a CONTINUE handler, the stored procedure continued the execution. As the result, we get the tag count for the article as well.

Example (Cont..)



USER Defined exception example

```
CREATE PROCEDURE UpdateEmployeeSalary(
    IN emp_id INT,
    IN new_salary DECIMAL(10,2)
)
BEGIN
    -- Check for invalid salary
    IF new_salary < 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Error: Salary cannot be negative',
                MYSQL_ERRNO = 10001; -- Custom error number
    END IF;

    -- If valid, update salary
    UPDATE Employees
    SET salary = new_salary
    WHERE employee_id = emp_id;

END $$
```

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Cursors

Cursors

- To handle a result set inside a stored procedure, we use a cursor.
- A cursor allows us to iterate a set of rows returned by a query and process each row accordingly.
- The set of rows the cursor holds is referred to as the **active set**.

1. We can declare a cursor by using the DECLARE statement:

```
DECLARE cursor_name CURSOR FOR SELECT_statement;
```

- The cursor declaration must be after any variable declaration.
- A cursor must always be associated with a SELECT statement.

Cursors

2. Next, open the cursor by using the OPEN statement.

```
OPEN cursor_name;
```

3. Then, use the FETCH statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.

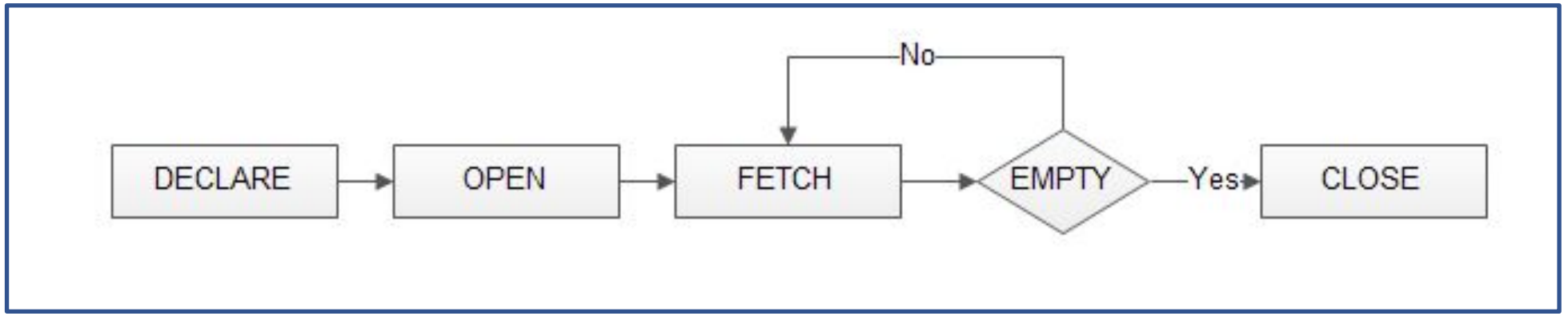
```
FETCH cursor_name INTO variables list;
```

4. Finally, call the CLOSE statement to deactivate the cursor and release the memory associated with it as follows:

```
CLOSE cursor_name;
```

Cursors

The following diagram illustrates how MySQL cursor works.



Example using Cursors

- **Example 2, updateSalary of Stored Procedure updates one row in deptsal table based on input parameter.**
- Suppose we want to update all the rows in deptsal simultaneously.
  - First, let's reset the totalsalary in deptsal to zero.

```
mysql> update deptsal set totalsalary = 0;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 3  Changed: 0  Warnings: 0

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |           0 |
|       2 |           0 |
|       3 |           0 |
+---------+-------------+
3 rows in set (0.00 sec)
```

Example using Cursors

```
mysql> delimiter $$
mysql> drop procedure if exists updateSalary$$
Query OK, 0 rows affected (0.00 sec)

mysql> create procedure updateSalary()
    -> begin
    ->          declare done int default 0;
    ->          declare current_dnum int;
    ->          declare dnumcur cursor for select dnumber from deptsal;
    ->          declare continue handler for not found set done = 1;
    ->
    ->          open dnumcur;
    ->
    ->          repeat
    ->                fetch dnumcur into current_dnum;
    ->                update deptsal
    ->                set totalsalary = (select sum(salary) from employee
    ->                                   where dno = current_dnum)
    ->                where dnumber = current_dnum;
    ->          until done
    ->          end repeat;
    ->
    ->          close dnumcur;
    -> end$$
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
```

Drop the old procedure

Use cursor to iterate the rows

Example using Cursors

- Call procedure :

```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |           0 |
|       2 |           0 |
|       3 |           0 |
+---------+-------------+
3 rows in set (0.01 sec)

mysql> call updateSalary;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

```
CREATE TABLE Employees (
    employee_id INT PRIMARY KEY,
    name VARCHAR(50)
);

CREATE TABLE Sales (
    sale_id INT PRIMARY KEY,
    employee_id INT,
    amount DECIMAL(10,2),
    FOREIGN KEY (employee_id)
REFERENCES
Employees(employee_id)
);
```

```
INSERT INTO Employees VALUES
(1, 'Alice'),
(2, 'Bob'),
(3, 'Charlie');

INSERT INTO Sales VALUES
(1, 1, 500),
(2, 1, 1000),
(3, 2, 2000),
(4, 3, 700),
(5, 3, 300);
```

```
CREATE PROCEDURE CalculateCommissions()
BEGIN
   DECLARE emp_id INT;
   DECLARE emp_name VARCHAR(50);
   DECLARE total_sales DECIMAL(10,2);
   DECLARE emp_cursor CURSOR FOR SELECT employee_id, name FROM Employees;
   DECLARE done INT DEFAULT 0;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
   OPEN emp_cursor;
   fetch_loop: LOOP
      FETCH emp_cursor INTO emp_id, emp_name;
      IF done = 1 THEN
         LEAVE fetch_loop;
      END IF;
      SELECT SUM(amount) INTO total_sales
      FROM Sales
      WHERE employee_id = emp_id;
SELECT CONCAT('Employee: ', emp_name,
       ' | Commission: $', IFNULL(total_sales,0) * 0.1) AS result;    END LOOP;
   CLOSE emp_cursor;
END $$
```

Calling the stored procedure

- CALL CalculateCommissions();

Employee: Alice | Commission: $150

Employee: Bob | Commission: $200

Employee: Charlie | Commission: $100

MIT-WPU
।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

Triggers

Triggers

- A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database i.e. when changes are made to the table.
- To monitor a database and take a corrective action when a condition occurs

Examples:

  - Charge $10 overdraft fee if the balance of an account after a withdrawal transaction is less than $500
  - Limit the salary increase of an employee to no more than 5% raise

- SQL triggers provide an alternative way to check the integrity of data.

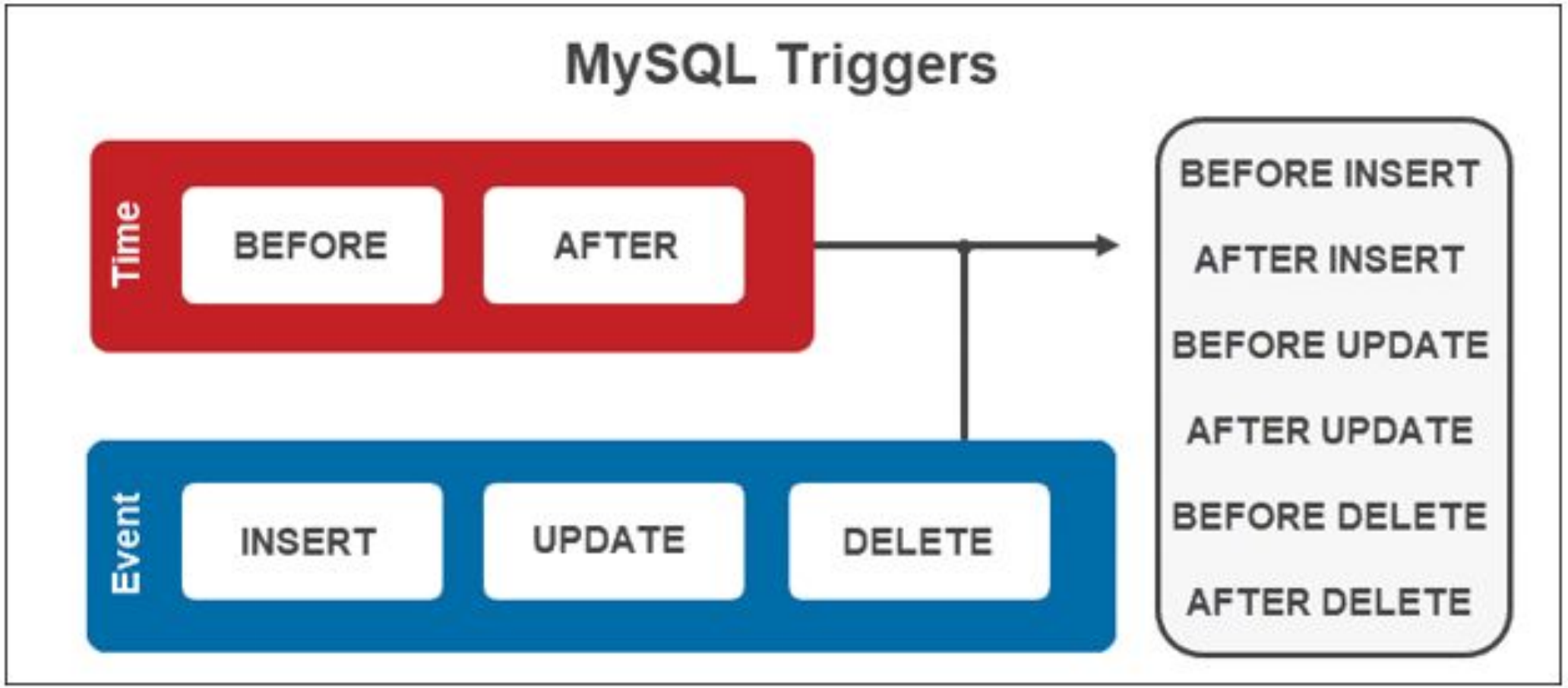
Triggering Events and Actions in SQL

- A trigger can be defined to be invoked either before or after the data is changed by **INSERT, UPDATE or DELETE .**
- MySQL allows you to define maximum six triggers for each table.
  - BEFORE INSERT – activated before data is inserted into the table.
  - AFTER INSERT- activated after data is inserted into the table.
  - BEFORE UPDATE – activated before data in the table is updated.
  - AFTER UPDATE - activated after data in the table is updated.
  - BEFORE DELETE – activated before data is removed from the table.
  - AFTER DELETE – activated after data is removed from the table.

MySQL Trigger Syntax

```
1 CREATE TRIGGER trigger_name trigger_time trigger_event
2  ON table_name
3  FOR EACH ROW
4  BEGIN
5  ...
6  END;
```

Trigger Time and Event



MySQL Trigger Example 1

- Create a BEFORE UPDATE trigger that is invoked before a change is made to the employees table.
- we used the OLD keyword to access employeeNumber and lastname column of the row affected by the trigger.

```
1  DELIMITER $$
2  CREATE TRIGGER before_employee_update
3      BEFORE UPDATE ON employees
4      FOR EACH ROW
5  BEGIN
6      INSERT INTO employees_audit
7      SET action = 'update',
8       employeeNumber = OLD.employeeNumber,
9          lastname = OLD.lastname,
10         changedat = NOW();
11 END$$
12 DELIMITER ;
```

Continued…

- In a trigger defined for INSERT, you can use NEW keyword only. You cannot use the OLD keyword.
- However, in the trigger defined for DELETE, there is no new row so you can use the OLD keyword only.
- In the UPDATE trigger, OLD refers to the row before it is updated and NEW refers to the row after it is updated.

Example (Cont..)

- Update the employees table to check whether the trigger is invoked.

```
1 UPDATE employees
2 SET
3     lastName = 'Phan'
4 WHERE
5     employeeNumber = 1056;
```

- Finally, to check if the trigger was invoked by the UPDATE statement, we can query the employees_audit table using the following query:

```
1 SELECT
2     *
3 FROM
4     employees_audit;
```

Example (Cont..)

- The following is the output of the query:

	id	employeeNumber	lastname	changedat	action
▶	1	1056	Phan	2015-11-14 21:39:12	update

Example 2

```
mysql> select * from employee;
+----+-------+----------+--------+--------------+-------+
| id | name  | superid  | salary | bdate        | dno   |
+----+-------+----------+--------+--------------+-------+
|  1 | john  |        3 | 100000 | 1960-01-01   |     1 |
|  2 | mary  |        3 |  50000 | 1964-12-01   |     3 |
|  3 | bob   |     NULL |  80000 | 1974-02-07   |     3 |
|  4 | tom   |        1 |  50000 | 1970-01-17   |     2 |
|  5 | bill  |     NULL |   NULL | 1985-01-20   |     1 |
+----+-------+----------+--------+--------------+-------+
5 rows in set (0.00 sec)

mysql> select * from deptsal;
+---------+--------------+
| dnumber | totalsalary  |
+---------+--------------+
|       1 |       100000 |
|       2 |        50000 |
|       3 |       130000 |
+---------+--------------+
3 rows in set (0.00 sec)
```

- We want to create a trigger to update the total salary of a department when a new employee is hired

Example 2 (Cont..)

Create a trigger to update the total salary of a department when a new employee is hired:

```
mysql> delimiter !
mysql> create trigger update_salary
    -> after insert on employee
    -> for each row
    -> begin
    ->         if new.dno is not null then
    ->             update deptsal
    ->             set totalsalary = totalsalary + new.salary
    ->             where dnumber = new.dno;
    ->         end if;
    -> end !
Query OK, 0 rows affected (0.06 sec)

mysql> delimiter ;
```

- The keyword “new” refers to the new row inserted

Example 2 (Cont..)

```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> insert into employee values (6,'lucy',null,90000,'1981-01-01',1);
Query OK, 1 row affected (0.08 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      190000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> insert into employee values (7,'george',null,45000,'1971-11-11',null);
Query OK, 1 row affected (0.02 sec)

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      190000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)

mysql> drop trigger update_salary;
Query OK, 0 rows affected (0.00 sec)
```

totalsalary increases by 90K

totalsalary did not change

MySQL Trigger

- To list all the triggers we have created:

mysql> show triggers;

```
1  SHOW TRIGGERS;
```

Trigger	Event	Table	Statement	Timing	Created	sql_mode	Definer
before_employee_update	UPDATE	employees	BEGIN INSERT INTO employ...	BEFORE	2015-11-14 21:39:09.08	STRICT_TRANS_TABLES,NO_AUTO_CREATE_U...	root@localhost

- To drop a trigger

mysql> drop trigger <trigger name>

http://www.mysqltutorial.org/

After Insert Example

```
mysql> delimiter //
mysql> CREATE TRIGGER person_ai AFTER INSERT
    -> ON person
    -> FOR EACH ROW
    -> UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
Query OK, 0 rows affected (0.30 sec)

mysql> delimiter ;
```

```
mysql> INSERT INTO person VALUES ('John', 19);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM person;
+---------+------+
| name    | age  |
+---------+------+
| Matthew |   25 |
| Mark    |   20 |
| John    |   19 |
+---------+------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM average_age;
+--------------+
| average      |
+--------------+
| 21.333333333 |
+--------------+
1 row in set (0.00 sec)
```

Before Update Example

```
mysql> delimiter ;
mysql> delimiter //
mysql> CREATE TRIGGER person_bu BEFORE UPDATE
    -> ON person
    -> FOR EACH ROW
    -> IF NEW.age < 18 THEN
    -> SIGNAL SQLSTATE '50002' SET MESSAGE_TEXT = 'Person must be older than 18';
    -> END IF; //
Query OK, 0 rows affected (0.36 sec)

mysql> delimiter ;
```

```
mysql> UPDATE person SET age = 17 WHERE name = 'John';
ERROR 1644 (50001): Person must be over the age of 18.
```

After Update example

```
mysql> delimiter //
mysql> CREATE TRIGGER person_au AFTER UPDATE
    -> ON person
    -> FOR EACH ROW
    -> UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
Query OK, 0 rows affected (0.93 sec)

mysql> delimiter ;
```

```
mysql> UPDATE person SET age = 21 WHERE name = 'John';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM person;
+---------+------+
| name    | age  |
+---------+------+
| Matthew |   25 |
| Mark    |   20 |
| John    |   21 |
+---------+------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM average_age;
+---------+
| average |
+---------+
|      22 |
+---------+
1 row in set (0.00 sec)
```

Before Delete

```
mysql> delimiter //
mysql> CREATE TRIGGER person_bd BEFORE DELETE
    -> ON person
    -> FOR EACH ROW
    -> INSERT INTO person_archive (name, age)
    -> VALUES (OLD.name, OLD.age); //
Query OK, 0 rows affected (0.33 sec)

mysql> delimiter ;
```

```
mysql> DELETE FROM person WHERE name = 'John';
Query OK, 1 row affected (0.03 sec)

mysql> SELECT * FROM person;
+---------+------+
| name    | age  |
+---------+------+
| Matthew |   25 |
| Mark    |   20 |
+---------+------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM person_archive;
+------+-----+---------------------+
| name | age | time                |
+------+-----+---------------------+
| John |  21 | 2021-04-09 09:37:57 |
+------+-----+---------------------+
1 row in set (0.00 sec)
```

After Delete example

```
mysql> delimiter //
mysql> CREATE TRIGGER person_ad AFTER DELETE
    -> ON person
    -> FOR EACH ROW
    -> UPDATE average_age SET average = (SELECT AVG(age) FROM person); //
Query OK, 0 rows affected (0.25 sec)

mysql> delimiter ;
```

```
mysql> DELETE FROM person WHERE name = 'John';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM person;
+---------+------+
| name    | age  |
+---------+------+
| Matthew |   25 |
| Mark    |   20 |
+---------+------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM average_age;
+---------+
| average |
+---------+
|    22.5 |
+---------+
1 row in set (0.00 sec)
```

Trigger for data integrity

```
CREATE TRIGGER prevent_negative_stock
BEFORE INSERT OR UPDATE ON inventory
FOR EACH ROW
BEGIN
    IF NEW.quantity < 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Quantity cannot be negative';
    END IF;
END;
```

Maintain referential integrity

```
CREATE TRIGGER delete_orders
AFTER DELETE ON customers
FOR EACH ROW
DELETE FROM orders WHERE customer_id = OLD.customer_id;
```

Applying Business Rules

```
CREATE TRIGGER apply_discount
BEFORE INSERT OR UPDATE ON sales
FOR EACH ROW
BEGIN
    IF NEW.total_amount > 1000 THEN
        NEW.discount = NEW.total_amount * 0.1;
    ELSE
        NEW.discount = 0;
    END IF;
END;
```

References

1. Silberschatz–Korth–Sudarshan's Database System Concepts, Seventh Edition.
2. MySQL Tutorial
   - http://www.mysqltutorial.org/