CSC 443 – Database Management Systems

Lecture 11 – SQL Procedures and Triggers

The SQL Programming Language

- By embedding SQL in programs written in other high-level programming languages, we produce ***impedance mismatch***
  - Mixing different programming paradigms
  - SQL is a declarative language
  - High-level language such as C is a procedural language
  - SQL and 3GLs use different models to represent data

SQL Programming Language (continued)

- SQL/PSM (Persistent Stored Modules)
- PL/SQL (Procedural Language/SQL)
  - Oracle's procedural extension to SQL
  - Two versions
- Many of these features also exist in MySQL.

Declarations

- Variables and constant variables must be declared before they can be referenced
- Examples
  - vStaffNo   VARCHAR(25);
  - vRent   NUMBER(6, 2) NOT NULL := 600
- Possible to declare a variable as NOT NULL
- %TYPE and %ROWTYPE

Declarations (continued)

- %TYPE and %ROWTYPE
- Examples – Individual variables
  – vStaffNo Staff.staffNo%TYPE;
  – vStaffNo1 vStaffNo%TYPE;
- Example – Row as a Variable
  – vStaffRec Staff%ROWTYPE;
- %TYPE and %ROWTYPE – are not standard SQL.

General Structure of a SQL Procedure Block

`[DECLARE`	*Optional*
`  --declarations]`	
`BEGIN`	*Mandatory*
`  -executable statements`	
`[EXCEPTION`	*Optional*
`  --exception handlers]`	
`END;`	*Mandatory*

Assignments

- Variables can be assigned in two ways:
  - Using the normal assignment statement (:)
  - Using an SQL SELECT or FETCH state
- Examples
  - **`vStaffNo :=‘SG14’;`**
  - **`vRent := 500;`**

  -
    ```
    SELECT COUNT(*) INTO x
      FROM PropertyForRent
      WHERE staffNo=vStaffNo;
    ```

Control Statements

- Conditional **`IF`** statement
- Conditional **`CASE`** statement
- Iteration statement (**`LOOP`**)
- Iteration statement (**`WHILE`** and **`REPEAT`**)

Conditional **IF** statement

```
IF search_condition
  THEN statement_list
     [ELSEIF search_condition
      THEN statement_list] ...
     [ELSE statement_list]
END IF
```

Conditional **IF** statement - Example

```
DELIMITER //
CREATE FUNCTION SimpleCompare(n INT, m INT)
  RETURNS VARCHAR(20)
  BEGIN
    DECLARE s VARCHAR(20);

    IF n > m THEN SET s = '>';
    ELSEIF n = m THEN SET s = '=';
    ELSE SET s = '<';
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m);
    RETURN s;
  END //
```

Conditional **CASE** statement

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list]
...
    [ELSE statement_list]
END CASE
```

Conditional **CASE** statement – Alternate Form

```
CASE
    WHEN search_condition
          THEN statement_list
    [WHEN search_condition
          THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Conditional **`CASE`** statement – Example

```
DELIMITER |
CREATE PROCEDURE p()
  BEGIN
    DECLARE v INT DEFAULT 1;
    CASE v
      WHEN 2 THEN SELECT v;
      WHEN 3 THEN SELECT 0;
      ELSE
        BEGIN
        END;
    END CASE;
  END;
  |
```

Iteration statement (**`LOOP`**)

- Syntax:

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

- The statements within the loop are repeated until the loop is terminated. Usually, this is accomplished with a **`LEAVE`** statement. Within a stored function, **`RETURN`** can also be used, which exits the function entirely.

LOOP Statement - Example

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN
      ITERATE label1;
    END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END;
```

Iteration statement (**WHILE**)

- Syntax:

```
[begin_label:] WHILE search_condition
    DO
     statement_list
    END WHILE [end_label]
```

- The statement list within a **WHILE** statement is repeated as long as the search_condition expression is true.
- **statement_list** consists of one or more SQL statements, each ending with a semicolon (;)

WHILE Statement - Example

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END;
```

Iteration statement (**REPEAT**)

- Syntax:

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

- The statement list within a **REPEAT** statement is repeated until the search_condition expression is true; a **REPEAT** always enters the loop at least once

REPEAT Statement - Example

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT
    ->     SET @x = @x + 1;
    ->   UNTIL @x > p1 END REPEAT;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
```

```
mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

Exceptions in PL/SQL

- Exception
  - Identifier in PL/SQL
  - Raised during the execution of a block
  - Terminates block’s main body of actions
- Exception handlers
  - Separate routines that handle raised exceptions
- User-defined exception
  - Defined in the declarative part of a PL/SQL block

Example of Exception Handling in PL/SQL

```
DECLARE
  vpCount   NUMBER;
  vStaffNo PropertyForRent.staffNo%TYPE := 'SG14';
--define an exception for the enterprise constraint
--that prevents a member of staff from managing more
--than 100 properties
  e_too_many_properties EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_too_many_properties, -
20000);
BEGIN
      SELECT COUNT(*) INTO VPCount;
      FROM PropertyForRent
      WHERE staffNo = vStaffNo;
      IF vpCount = 100
--raise an exception for the general constraint
      RAISE e_too_many_properties;
      END IF;
      UPDATE PropertyForRentset staffNo = vStaffNo
WHERE propertyNo='PG4';
EXCEPTION
--handle the exception for the general constraint
  WHEN e_too_many_properties THEN
      dbms_output.put_line('Member of staff' ||
staffNo || 'already managing 100 properties');
END;
```

Condition Handling

- Define a handler by
  - Specifying its type
  - Exception and completion conditions it can resolve
  - Action it takes to do so
- Handler is activated
  - When it is the most appropriate handler for the condition that has been raised by the SQL statement

The **DECLARE . . . HANDLER** Statement

```
DECLARE handler_action HANDLER
FOR condition_value [, condition_value] ...
statement

handler_action: CONTINUE | EXIT
              | UNDO  (not supported by MySQL)

condition_value:  mysql_error_code |
                  SQLSTATE [VALUE] sqlstate_value |
                  condition_name |
                  SQLWARNING |
                  NOT FOUND | SQLEXCEPTION
```

Cursors in SQL

- Cursor
  - Allows the rows of a query result to be accessed one at a time
  - Must be declared and opened before use
  - Must be closed to deactivate it after it is no longer required
  - Updating rows through a cursor

Using Cursors in PL/SQL to Process a Multirow Query

```
DECLARE
  vPropertyNo     PropertyForRent.propertyNo%TYPE;
  vStreet   PropertyForRent.street%TYPE;
  vCity           PropertyForRent.city%TYPE;
  vPostcode PropertyForRent.postcode%TYPE;
  CURSOR propertyCursor IS
         SELECT propertyNo,street, city, postcode
         FROM PropertyForRent
         WHERE staffNo = 'SG14'
         ORDER BY propertyNo;
```

```
BEGIN
--Open the cursor to start of seelction, then loop
--to fetch each row of the result table.
  OPEN propertyCursor;
  LOOP

--Fetch next row of the result table
  FETCH propertyCursor
    INTO vPropertyNo, vStreet, vCity, vPostcode;
  EXIT WHEN propertyCursor%NOTFOUND;

--Display data
      dbms_output.put_line
         ('Property number:' ||vPropertyNo);
      dbms_output.put_line
         ('Street       '||vStreet);
      dbms_output.put_line('City          '||vCity);
      IF postcode IS NOT NULL THEN
        dbms_output.put_line
           ('Postal Code  '||vPostcode);
      ELSE
        dbms_output.put_line('Postal Code  NULL');
      END IF;
  END LOOP;
  IF propertyCursor%ISOPEN
     THEN CLOSE propertyCursor
  END IF;
```

```
--Error condition print out error
EXCEPTION
  WHEN OTHER THEN
      dbms_output.put_line('Error detected');
        IF propertyCursor%ISOPEN THEN CLOSE
propertyCursor END IF:
END;
```

Subprograms, Stored Procedures, Functions, and Packages

- Package
  - Collection of procedures, functions, variables, and SQL statements that are grouped together and stored as a single program unit
  - Specification
    - Declares all public constructs of the package
  - Body
    - Defines all constructs (public and private) of the package

Subprograms, Stored Procedures, Functions, and Packages (continued)

- Subprograms
  - Named PL/SQL blocks that can take parameters and be invoked
  - Types
    - Stored procedures
    - Functions
  - Parameters

Triggers

- Trigger
  - Defines an action that the database should take when some event occurs in the application
  - Format of a trigger
  - Types
  - TRIGGER Privilege
  - Advantages and disadvantages of triggers

`CREATE TRIGGER` Syntax

```
CREATE [DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name
trigger_time trigger_event
ON tbl_name FOR EACH ROW
trigger_body

trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
```