Database Systems 2

Lecture 8

PL/SQL and Triggers

Lecture - Objectives

PL/SQL Overview

Language features

Triggers

PL/SQL

Procedural Language / Structured Query Language

SQL is a declarative query language, not a full procedural programming language, although it does have a CREATE FUNCTION statement

PL/SQL is an extension to Oracle SQL which provides procedural language features.

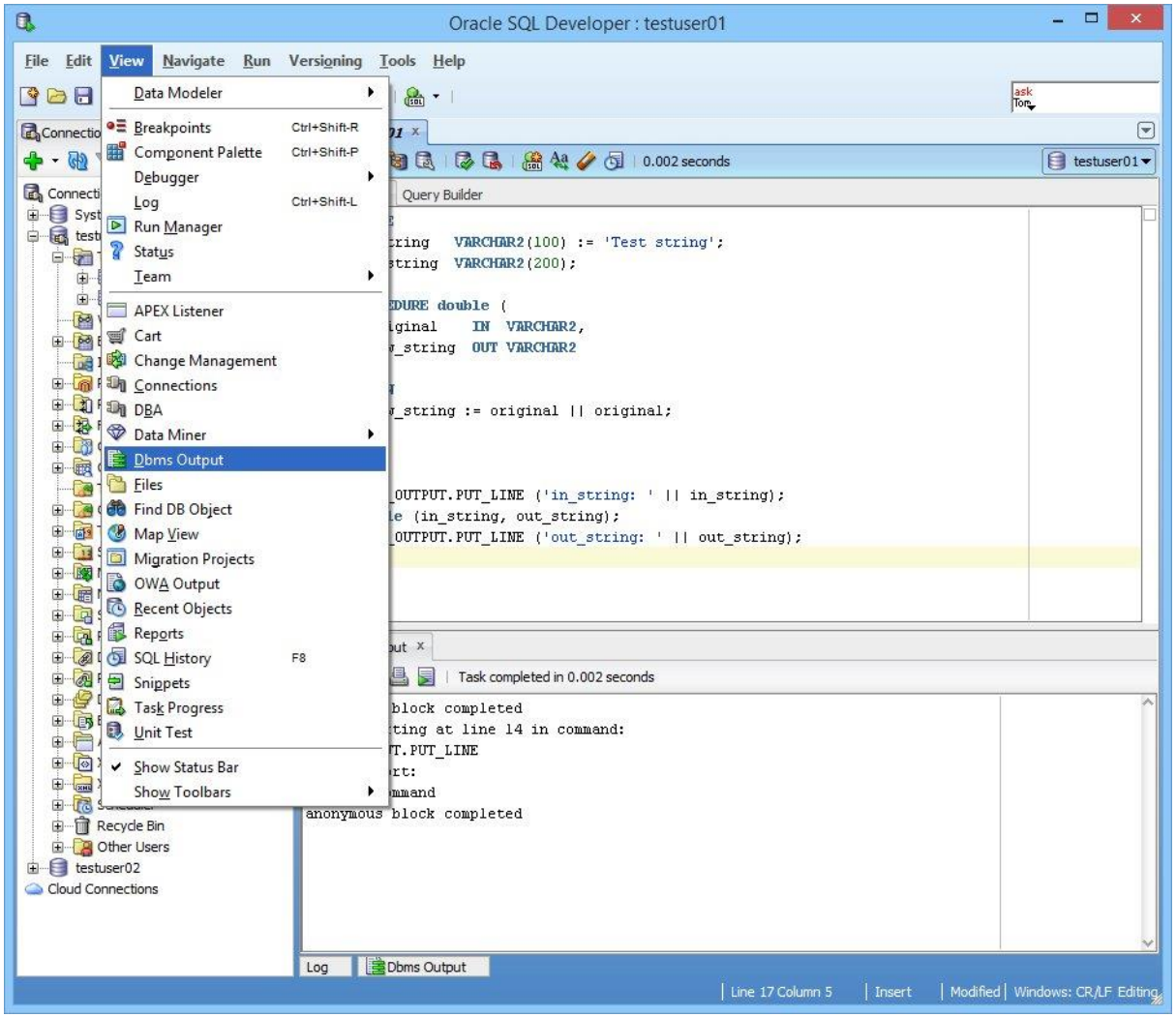
Other Database products often have a similar procedural language extensions.

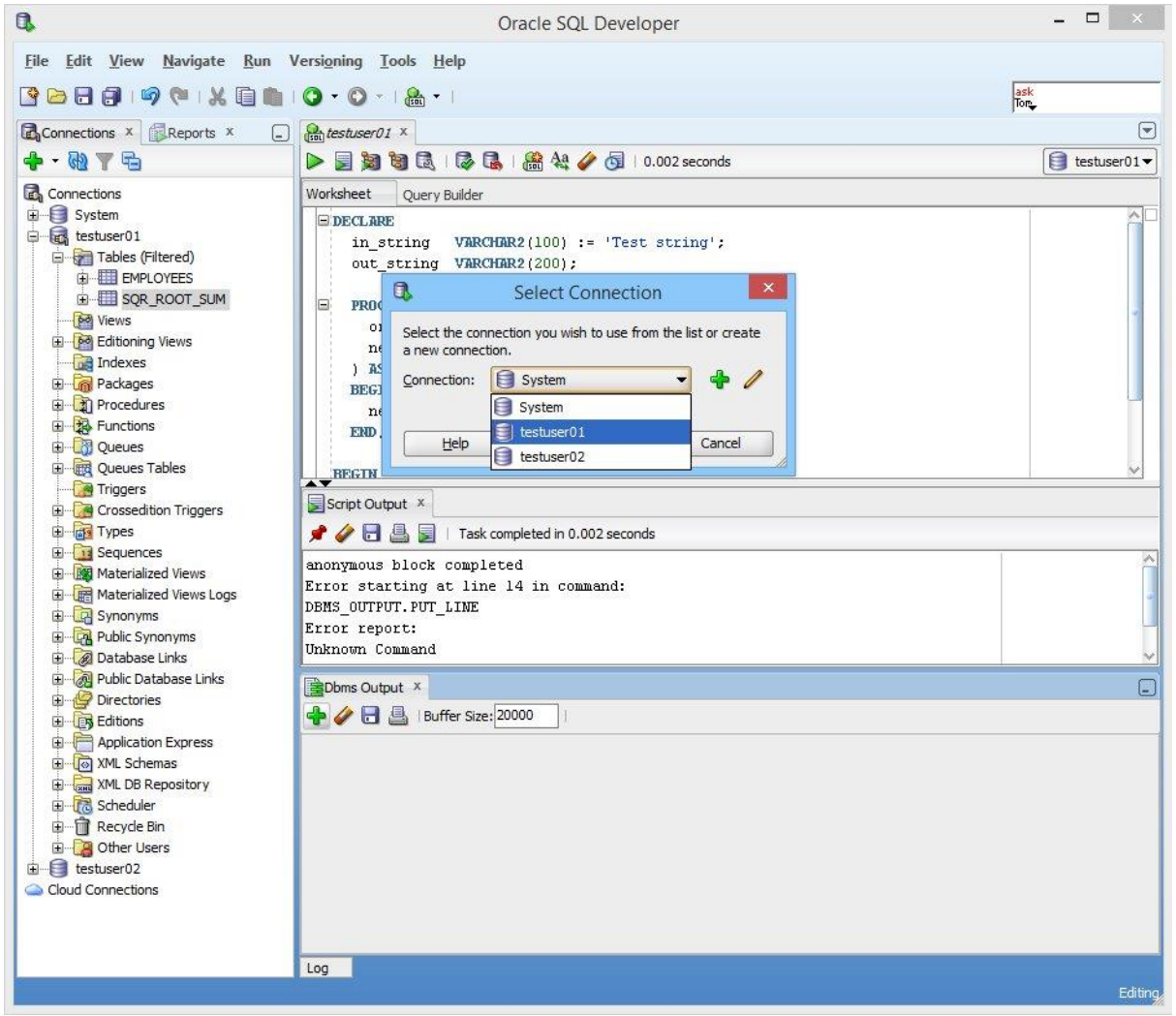
When Using the CLI

To get the error messages displayed on the command line, type in:

Set ServerOutput On

Will direct DBMS_OUTPUT to the CLI window for the rest of the session.


Oracle SQL Developer : testuser01
File Edit View Navigate Run Versioning Tools Help
Data Modeler
Breakpoints Ctrl+Shift-R
Component Palette Ctrl+Shift-P
Debugger
Log Ctrl+Shift-L
Run Manager
Status
Team
APEX Listener
Cart
Change Management
Connections
DBA
Data Miner
Dbms Output
Files
Find DB Object
Map View
Migration Projects
OWA Output
Recent Objects
Reports
SQL History F8
Snippets
Task Progress
Unit Test
Show Status Bar
Show Toolbars
0.002 seconds
testuser01
Query Builder
VARCHAR2(100) := 'Test string';
VARCHAR2(200);
DURE double (
IN VARCHAR2,
OUT VARCHAR2
_string := original || original;
OUTPUT.PUT_LINE ('in_string: ' || in_string);
(in_string, out_string);
OUTPUT.PUT_LINE ('out_string: ' || out_string);
Task completed in 0.002 seconds
block completed
ting at line 14 in command:
T.PUT_LINE
rt:
mmand
anonymous block completed
Recycle Bin
Other Users
testuser02
Cloud Connections
Log
Dbms Output
Line 17 Column 5
Insert
Modified
Windows: CR/LF
Editing


Oracle SQL Developer
File Edit View Navigate Run Versioning Tools Help
Connections
Reports
testuser01
0.002 seconds
Worksheet
Query Builder
DECLARE
in_string VARCHAR2(100) := 'Test string';
out_string VARCHAR2(200);
Select Connection
Select the connection you wish to use from the list or create a new connection.
Connection:
System
testuser01
testuser02
Help
Cancel
Script Output
Task completed in 0.002 seconds
anonymous block completed
Error starting at line 14 in command:
DBMS_OUTPUT.PUT_LINE
Error report:
Unknown Command
Dbms Output
Buffer Size: 20000
Log
Editing
Connections
System
testuser01
Tables (Filtered)
EMPLOYEES
SQR_ROOT_SUM
Views
Editioning Views
Indexes
Packages
Procedures
Functions
Queues
Queues Tables
Triggers
Crossedition Triggers
Types
Sequences
Materialized Views
Materialized Views Logs
Synonyms
Public Synonyms
Database Links
Public Database Links
Directories
Editions
Application Express
XML Schemas
XML DB Repository
Scheduler
Recycle Bin
Other Users
testuser02
Cloud Connections

Procedural Language Features

- Blocks
- Variables
- Input and Output statements
- Control structures
- Conditions
  - Iteration
  - Sequential
  - Subprograms
- Exceptions (Run time error handling)

Basic Structure of a PL/SQL Block

```
DECLARE    -- [optional]
  -- Declarations of variables

BEGIN      -- Executable part (required)
  -- Statements

EXCEPTION -- [optional]
  -- Exception handlers to catch errors

END;
```

Using SELECT INTO to Assign Values to Variables

```
CREATE TABLE employees
(
employee_id     NUMBER(6),
salary          NUMBER(8,2),
job_id          VARCHAR(8),
PRIMARY KEY (employee_id)
);

INSERT INTO employees VALUES (100, 200.50, 'PU_CLERK');
INSERT INTO employees VALUES (115, 200.50, 'PU_CLERK');
INSERT INTO employees VALUES (200, 400.50, 'PU_CLERK');
```

Use of PL SQL to Assign Values to Variables

```
DECLARE
  bonus   NUMBER(8,2);
BEGIN
  SELECT salary * 0.10 INTO bonus
   FROM employees
     WHERE employee_id = 100;
  DBMS_OUTPUT.PUT_LINE('The bonus for employee 100 is: ' || bonus);
END;
```

This will place the code into the buffer, but it will not be executed until you enter the forward slash.

```
/
```

```
DECLARE
   job_var         employees.job_id%TYPE;
   sal_var         employees.salary%TYPE;
   sal_raise       NUMBER(3,2);

BEGIN
  SELECT job_id, salary INTO job_var, sal_var
    FROM employees
      WHERE employee_id = 115;

  CASE
    WHEN job_var = 'PU_CLERK' THEN
      IF sal_var < 3000 THEN
        sal_raise := 0.12;
      ELSE
        sal_raise := 0.09;
      END IF;

    ELSE
      BEGIN
        DBMS_OUTPUT.PUT_LINE('No raise for this job: ' || job_var);
      END;
   END CASE;

   UPDATE employees
     SET salary = salary + salary * sal_raise
       WHERE employee_id = 115;
END;
```

Use of IF THEN ELSE and CASE statement to UPDATE the data in a table

Note use of DBMS_OUTPUT.PUT_LINE

Using the For Loop

```
CREATE TABLE sqr_root_sum
 (
  num       NUMBER,
  sq_root   NUMBER(6,2),
  sqr       NUMBER,
  sum_sqrs  NUMBER
);
```

PL/SQL also has a While Loop

```
DECLARE
   s_var  PLS_INTEGER;
BEGIN
  FOR i in 1..100
    LOOP
      s_var := (i * (i + 1) * (2*i +1)) / 6;      -- sum of squares

      INSERT INTO sqr_root_sum VALUES (i, SQRT(i), i*i, s_var );
    END LOOP;
END;
```

PL/SQL Subprogram

```
DECLARE
  in_var   INTEGER(3) := 25;
  out_var  INTEGER(3);

  PROCEDURE double ( original IN INTEGER, new_var OUT INTEGER )
  AS
  BEGIN
    new_var := original + original;
  END;

BEGIN
  DBMS_OUTPUT.PUT_LINE ('in_string: ' || in_var);
  double (in_var, out_var);
  DBMS_OUTPUT.PUT_LINE ('out_string: ' || out_var);
END;
```

Other Types of PL/SQL Block

Anonymous

```
DECLARE


BEGIN
  -- statements


EXCEPTION

END;
```

Procedure

```
PROCEDURE name
IS


BEGIN
  -- statements


EXCEPTION

END;
```

Function

```
FUNCTION name
RETURN datatype
IS

BEGIN
  -- statements

  RETURN value;

EXCEPTION

END;
```

PL/SQL Blocks

An anonymous block is executed as soon as it is entered.

A named procedure block can be stored and executed repeatedly. They can be called from other procedures, functions and triggers within an application.

A named function is a procedure which returns a value.

Triggers

A trigger is a stored subprogram, which is associated with a table, view or event.

The trigger can be invoked once, when some event occurs, or many times, once for each row affected by an INSERT, UPDATE or DELETE statement.

The trigger can be invoked before or after the event.

CREATE TRIGGER Example

```
CREATE OR REPLACE TRIGGER Print_salary_changes
  BEFORE DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
DECLARE
    sal_diff    NUMBER;
BEGIN
    sal_diff  := :NEW.salary  - :OLD.salary;
    DBMS_OUTPUT.PUT_LINE(chr(10));
    DBMS_OUTPUT.PUT('Old salary: ' || :OLD.salary);
    DBMS_OUTPUT.PUT('  New salary: ' || :NEW.salary);
    DBMS_OUTPUT.PUT_LINE('  Difference ' || sal_diff);
END;
/
```

```
CREATE TABLE emp_audit
(
  emp_audit_id  NUMBER(6),
  up_date       DATE,
  new_sal       NUMBER(8,2),
  old_sal       NUMBER(8,2)
);
```

Another Example

Assuming the existence of an employee table, and an emp_audit table

```
CREATE TRIGGER audit_sal
  AFTER UPDATE OF salary
    ON employees
      FOR EACH ROW
BEGIN
  INSERT INTO emp_audit
    VALUES(:OLD.employee_id, SYSDATE, :NEW.salary, :OLD.salary);
END;
/
```

Overview of Triggers

A trigger is a named program unit that is stored in the database and fired (executed) in response to a specified event. The specified event is associated with either a table, a view, a schema, or the database, and it is one of the following:

A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)

A database definition (DDL) statement (CREATE, ALTER, or DROP)

A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN)

The trigger is said to be defined on the table, view, schema, or database.

Trigger Types

A DML trigger is fired by a DML statement, a DDL trigger is fired by a DDL statement, a DELETE trigger is fired by a DELETE statement, and so on.

An INSTEAD OF trigger is a DML trigger that is defined on a view (not a table). The database fires the INSTEAD OF trigger instead of executing the triggering DML statement. For more information, see Modifying Complex Views (INSTEAD OF Triggers).

A system trigger is defined on a schema or the database. A trigger defined on a schema fires for each event associated with the owner of the schema (the current user). A trigger defined on a database fires for each event associated with all users.

A simple trigger can fire at exactly one of the following timing points:

- Before the triggering statement executes
- After the triggering statement executes
- Before each row that the triggering statement affects
- After each row that the triggering statement affects

A compound trigger can fire at more than one timing point.

Uses of Triggers

Triggers supplement the standard capabilities of your database to provide a highly customized database management system. For example, you can use triggers to:

- Automatically generate derived column values
- Enforce referential integrity across nodes in a distributed database
- Enforce complex business rules
- Provide transparent event logging
- Provide auditing
- Maintain synchronous table replicates
- Gather statistics on table access
- Modify table data when DML statements are issued against views
- Restrict DML operations against a table to those issued during regular business hours
- Enforce security authorizations
- Prevent invalid transactions

Guidelines for Designing Triggers 1

Use triggers to guarantee that when a specific operation is performed, related actions are performed.

Do not define triggers that duplicate database features.

- For example, do not define triggers to reject bad data if you can do the same checking through constraints.
  - NOT NULL, UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK

Although you can use both triggers and integrity constraints to define and enforce any type of integrity rule, Oracle strongly recommends that you use triggers to constrain data input only in the following situations:

- To enforce referential integrity when child and parent tables are on different nodes of a distributed database

- To enforce complex business rules not definable using integrity constraints

Guidelines for Designing Triggers 2

Limit the size of triggers.

- If the logic for your trigger requires much more than 60 lines of PL/SQL code, put most of the code in a stored subprogram and invoke the subprogram from the trigger.
- The size of the trigger cannot exceed 32K.

Use triggers only for centralized, global operations that must fire for the triggering statement, regardless of which user or database application issues the statement.

Do not create recursive triggers.

- For example, if you create an AFTER UPDATE statement trigger on the employees table, and the trigger itself issues an UPDATE statement on the employees table, the trigger fires recursively until it runs out of memory.

Use triggers on DATABASE judiciously. They are executed for every user every time the event occurs on which the trigger is created.

Contraints and Triggers

Triggers and declarative constraints can both be used to constrain data input. However, triggers and constraints have significant differences.

Declarative constraints are statements about the database that are always true. A constraint applies to existing data in the table and any statement that manipulates the table.

Triggers constrain what a transaction can do. A trigger does not apply to data loaded before the definition of the trigger; therefore, it is not known if all data in a table conforms to the rules established by an associated trigger.

The following example should have been done by using a standard SQL constraint.

DELETE Cascade Trigger for Parent Table

```
CREATE TRIGGER Dept_del_cascade
  AFTER DELETE ON dept
    FOR EACH ROW
BEGIN
  DELETE FROM emp
    WHERE emp.Deptno = :OLD.Deptno;
END;
```

Should have been done by using this constraint on the foreign key:

```
FOREIGN KEY emp.deptno REFERENCES dept(deptno)
ON DELETE CASCADE
```

Trigger for Complex Check Constraints

Triggers can enforce integrity rules other than referential integrity. For example, this trigger performs a complex check before allowing the triggering statement to run.

Using the employees table again.

First I want to clear all the existing data out:

```
DELETE FROM employees;
```

Then insert one new record:

```
INSERT INTO employees VALUES (44, 300.00, 'PU_CLERK');
```

```
CREATE OR REPLACE TRIGGER Salary_check
     BEFORE INSERT OR UPDATE OF Salary ON employees
FOR EACH ROW
DECLARE
  Minsal               NUMBER(8,2) := 100.00;
  Maxsal               NUMBER(8,2) := 500.00;
  Salary_out_of_range  EXCEPTION;

BEGIN

/* If employee's new salary is less than or greater than
   job classification's limits, raise exception.
   Exception message is returned and pending INSERT or UPDATE statement
   that fired the trigger is rolled back:*/
```

```
 IF (:NEW.Salary < Minsal OR :NEW.Salary > Maxsal) THEN
           RAISE Salary_out_of_range;
  END IF;

EXCEPTION
  WHEN Salary_out_of_range THEN
      Raise_application_error (-20300, 'Salary ' ||TO_CHAR(:NEW.Salary)
             ||' out of range for employee '||TO_CHAR(:NEW.Employee_id));

  WHEN NO_DATA_FOUND THEN
      Raise_application_error(-20322,
        'Invalid Job Classification ' ||:NEW.Job_id);
  END;
/
```

Trigger the trigger by doing this:

```
UPDATE employees SET Salary = 1000.00;
```

See Oracle Docs

http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/toc.htm

http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/triggers.htm#autoId54