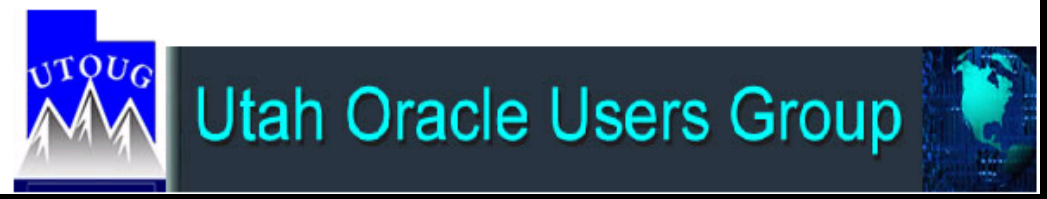


PL/SQL Concepts

– for Beginners

Oracle Training

Featuring
Sam Weber, OCP
Senior DBA, Attask Inc.

After taking this class, you should know how to:

- define PL/SQL
- Learn and understand the use of the 3 Main PL/SQL block sections
- Create and execute an anonymous PL/SQL block from SQL*Plus
- Create and call a named PL/SQL block as a stored object in an Oracle schema

After taking this class, you should know how to:

- Handle exceptions and error messages in your PL/SQL
- Understand the uses of (and differences between) stored procedures, functions, triggers and packages
- Use If-then statements to branch your coding logic
- Create and use simple For-next and While Loops to process multiple records

What is PL/SQL?

Procedural Language/Structured Query Language is Oracle's own language developed to code stored procedures that interacts seamlessly with database objects…

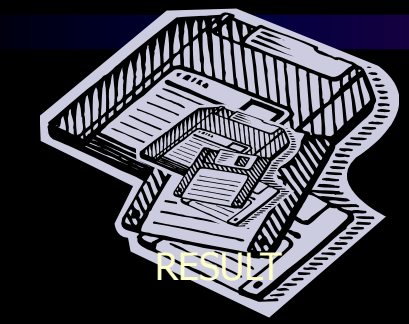
PL/SQL is similar to other coding languages such as C++ or Java in that it allows the use of declared variables, IF-THEN branching, LOOPS and advanced error handling

PL/SQL much more powerful than standard SQL And can be recompiled into native languages.

Why use PL/SQL?

- Communicates natively with DB objects - faster on queries and DML
- Managed as objects in the DB - easier to maintain and edit
- Operates directly on the Server itself which tends to be a faster box with more RAM
- Broad base of features and functions allow far more flexible uses than SQL

Regular Code - Processed on the Client

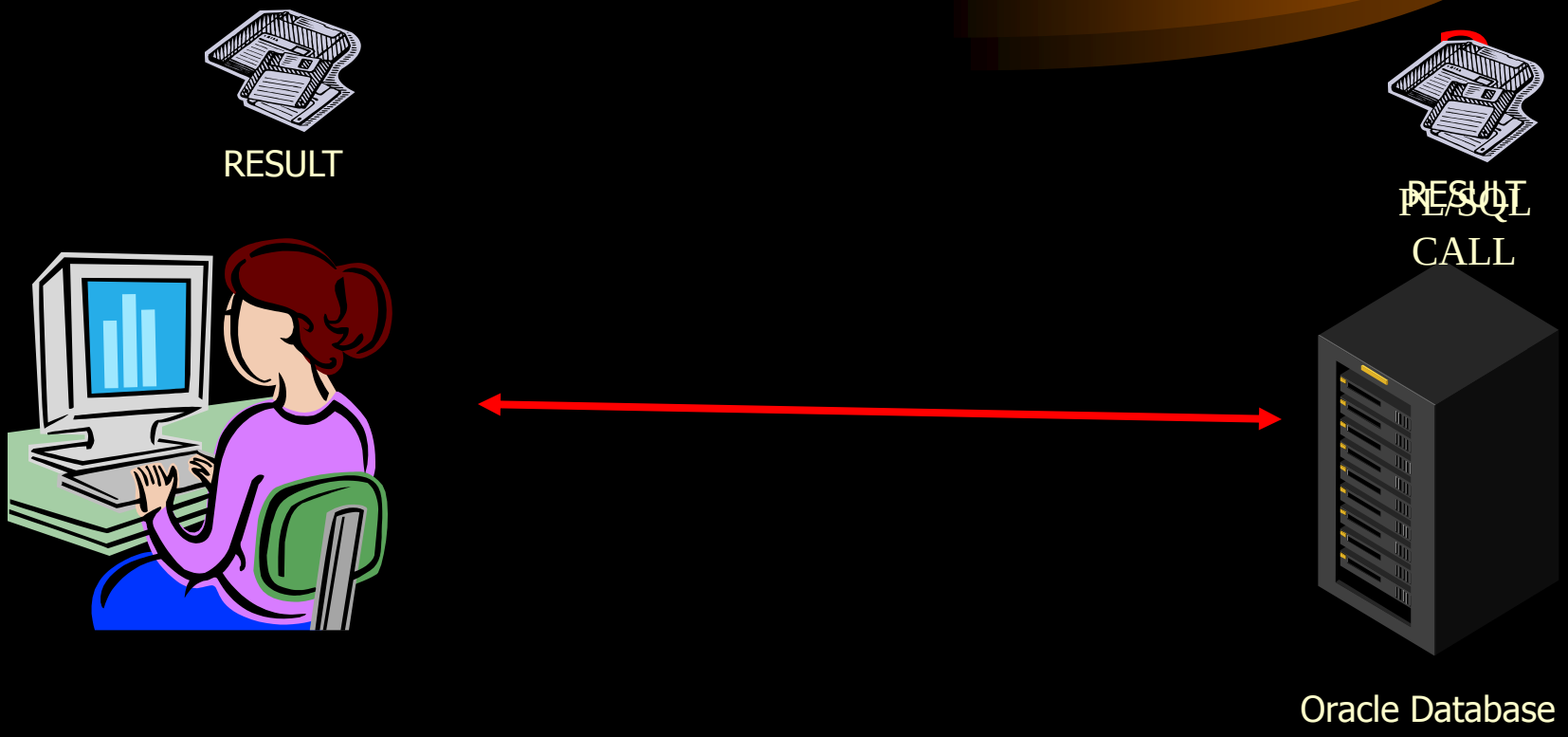


DATA

DATA

Oracle Database

PL/SQL - Process on the Server



The PL/SQL Block Structure
the 3 main blocks

- Variable declaration section - optionally may begin with DECLARE - identifies all variable names and init values
- Executable - mandatory - starts with BEGIN, ends with END: - contains executing code
- Exception handler - optional - starts with keyword EXCEPTION inside Exec section

The PL/SQL Block

DECLARATION AREA

EXECUTION AREA

EXCEPTION AREA

```
v_Counter   Number := 0;
V_Title     Varchar2(25);
BEGIN
  select count(*) from movie into v_Counter
  where  title = v_Title;
EXCEPTION
WHEN NO_DATA_FOUND THEN RAISE
END;
```

DECLARATION section

- Cursors, Variables and Arrays declared
- Optional
- Doesn’t require explicit DECLARE
- data type can be hard-coded or typed by table_name.column_name

Variables

- The name of an address in memory
- Content may be passed IN or OUT
- Assignment done by using: :=
- data type can be hard-coded or typed by table_name.column_name
- Can be used in equations, functions and predicate comparisons in SQL statements

DECLARE *examples*

```
n_Counter    Number := 0;
v_Type       Varchar2(10);
v_Boolean    Char(1) := 'T';
V_Title MOVIE.TITLE%TYPE;
```

The **Executable** Section

- Stores all the statements processed by Oracle when the block is ran.
- Each executable statement must be terminated by a semi-colon.
- May use same operators, predicates and functions that SQL uses.
- Additionally, may use variables and programming formulas/constructs.

Exception Handling

- Any Oracle error can be mapped to an exception and RAISED
- Exception block is optional
- Can print using DBMS_OUTPUT:
- Special Code Blocks can be executed in the event of specific errors
- Create your own error messages

Exception Handling *Example*

```
EXCEPTION
            WHEN NO_DATA_FOUND THEN
NULL;
            WHEN OTHERS THEN RAISE;
```

WHEN OTHERS THEN DBMS_OUTPUT.Put_line (SQLCODE || SQLERRM);

SET Serveroutput on size 10000

- SQL*Plus Set environment command
- Enables screen display within PL/SQL code
- Size parameter defines the amount of buffer available to store output
- Size parameter is optional
- Used with DBMS_OUTPUT.PUT_LINE

What is DBMS_OUTPUT.PUT_LINE?

- One of hundreds of PL/SQL packages (stored objects) provided by Oracle in it's internal data dictionary
- Can be called like any other stored object
- Display hard coded text and values stored in memory to the screen
- Arguments are passed to the package

"Hello World" Example

```
SET SERVEROUTPUT ON SIZE 10000

BEGIN
DBMS_OUTPUT.PUT_LINE ('HELLO WORLD!');
END;
/
```

Task

- Add a 2nd line to your ‘Hello World’ example displaying the words ‘My name is ….’

1a

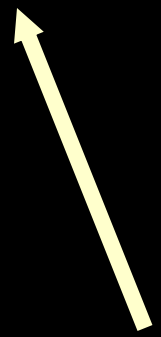
Solution!

```
SET SERVEROUTPUT ON SIZE 10000

BEGIN
DBMS_OUTPUT.PUT_LINE ('HELLO WORLD!
My name is Sam');
END;
/
```

BEGIN & END – DML example

```
BEGIN
INSERT INTO MOVIE (TITLE) VALUES
('LORD OF THE RINGS-RETURN OF THE KING');
COMMIT;
END;
/
```



ANY DML SQL WILL WORK HERE

Add a Declare Section

```
DECLARE
v_movie varchar2(100);
BEGIN
v_Movie := 'Lord of the Rings – Return of the King';
INSERT INTO MOVIE (TITLE) VALUES (v_movie);
COMMIT;
END;
/
```

Now we add a variable to pass values in

Concatenation in PUT_LINE

The double bar (||) is used in SQL to blend strings with other strings OR with a column name. PL/SQL adds the ability to display the value of a variable in memory

DBMS_OUTPUT.PUT_LINE can use this to display a message:

DBMS_OUTPUT.PUT_LINE (‘Movie Name is ‘ || v_movie);

TASK!

- Edit the insert a movie PL/SQL to insert your favorite movie
- Add a DBMS_OUTPUT.PUT_LINE display to the screen a message showing your favorite movie was added successfully to the movie table

3a

Solution!

```
DECLARE
v_movie varchar2(100);
BEGIN
v_Movie := 'Avatar';
INSERT INTO MOVIE (TITLE) VALUES (v_movie);
COMMIT;
DBMS_OUTPUT.PUT_LINE (v_Movie||' inserted
successfully');
END;
/
```

A PL/SQL block EXAMPLE

```
DECLARE
V_Counter number;
BEGIN
SELECT COUNT(*) INTO v_Counter
FROM MOVIE WHERE upper(TITLE) LIKE 'LORD%';
IF v_Counter <> 0 THEN
DBMS_OUTPUT.PUT_LINE ('There are '|| v_Counter || ' movies
starting with the word LORD');
END IF;
END;
/
```

Indirect Cursors

```
DECLARE
V_Counter number;
BEGIN
SELECT COUNT(*) INTO v_Counter
FROM MOVIE WHERE upper(TITLE) LIKE 'LORD%';
DBMS_OUTPUT.PUT_LINE ('There are '|| v_Counter || ' movies
starting with the word LORD');
END;
/
```

What is an indirect cursor?

- A single row select query
- Returns 1 or more fields
- Value is assigned to a named area of memory
- Named area is a PL/SQL variable
- Often used with aggregate functions like count, sum etc.

Indirect Cursor Example

```
DECLARE
V_TODAY_DATE DATE;
BEGIN
SELECT SYSDATE
INTO v_TODAY_DATE
FROM DUAL;
END;
/
```

Indirect Cursor Multiple Row Example

```
DECLARE
v_title varchar2(50);  v_star varchar2(50);
BEGIN
SELECT TITLE, STAR
INTO v_Title, v_Star
FROM Movie where ID = 245;
END;
/
```

What happens if you get no rows or more than 1?

- 0 Rows returns: ORA-01403: no data found
- Greater than 1 rows returned results in: ORA-01422 exact fetch returns more than requested number of rows
- Normally see it doesn't happen!
- If it does, PL/SQL can handle it

The Movie Table

TITLE	**VARCHAR2(50)**
TYPE	**VARCHAR2(25)**
RATING	**VARCHAR2(15)**
STAR	**VARCHAR2(25)**
ID	**NUMBER**
MOVIE_DATE	**DATE**

TASK!

- Create a PL/SQL program with an indirect cursor which assigns a count of how many movies are of type ‘Comedy’
- Display that number using DBMS_OUTPUT.PUT_LINE

4b

Solution!

```
DECLARE
V_Counter number;
BEGIN
SELECT COUNT(*) INTO v_Counter
FROM MOVIE WHERE upper(type) LIKE 'COMEDY
DBMS_OUTPUT.PUT_LINE
('Number of comedy movies: '|| v_Counter );
END;
/
```

PL/SQL Syntax and Constructs

- Assignment of Variables and Constants
- Boolean Operators
- Predicate Operators
- String Concatenation
- Arithmetic Expressions
- IF THEN statements
- LOOPs

Assignment of Variables and Constants

Assign values to variables

```
v1 := 100
date1 := SYSDATE + 7;
```

Initialize a variable

```
LordofRings varchar2 := ‘One of the Best Movies Ever’;
Avatar varchar2 := ‘good graphics but lousy political message ’;
```

Initialize a constant

```
pi CONSTANT number := 3.145;
```

Default Values

- Default of string values is NULL
  - An empty string is still NULL in Oracle
- Default of date is NULL
- Default of numeric is 0

Boolean Operators

LOGICAL AND, OR and NOT

```
  IF A < B
    AND
    NOT
(C = 100
    OR
 C= 150)
THEN D := 5000;
```

Predicate Operators

=	Equal
!= <>	Not Equal
>	Greater Than
>=	Greater Than or Equal
<	Less Than
<=;	Less Than or Equal

Predicate Operators

BETWEEN	between 100 and 200
IN	IN (‘A’,’B’,’Z’)
LIKE	LIKE ‘%New York%’
	LIKE ‘_Z%’

Arithmetic Expressions

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power of
( )	To group expressions

IF THEN

IF BOOLEAN expression THEN

Statement;

ELSIF BOOLEAN expression THEN

Statement;

ELSE

Statement;

END IF;

Counter

```
V_CNT number := 0
BEGIN
LOOP
----------------------------------
V_CNT := V_CNT + 1;
END LOOP;
```

IF THEN Example

```
IF      movie.type = ‘comedy’       THEN
Comedy_count := Comedy_count +1;
ELSIF       movie.type = ‘action’ THEN
Action_count := Action_count + 1;
ELSE
Other_count := Other_count + 1;
END IF;
```

IF ! THEN Example

```
IF  NOT          movie.type = ‘comedy’      THEN
Other_count := Other_count +1;
ELSE
Comedy_count := Comedy_count + 1;
END IF;
```

! Can be used instead of NOT

TASK!

- Using the PL/SQL program in task 4 – Add IF THEN logic to display different messages depending on the number of Comedy Movies found
- If number is less < 10 then display ‘Few comedy movies’ elsif > 10 and < 35 then display ‘Some Comedy movies’ else display ‘Many Comedy Movies’;
- Display the messages using DBMS_OUTPUT.PUT_LINE

5b

Solution!

```
DECLARE
V_Counter number;
BEGIN
SELECT COUNT(*) INTO v_Counter
FROM MOVIE WHERE upper(type) LIKE 'COMEDY';
IF v_Counter < 10 then
DBMS_OUTPUT.PUT_LINE ('Few comedy movies');
ELSIF v_Counter BETWEEN 11 AND 35 THEN
DBMS_OUTPUT.PUT_LINE ('Some comedy movies');
ELSE
DBMS_OUTPUT.PUT_LINE ('Many comedy movies');
END IF;
END;
/
```

Simple LOOP

```
LOOP
Statement;
  IF   BOOLEAN expression      THEN
  EXIT;
  END IF;
END LOOP;
```

Simple LOOP Example

```
LOOP
Insert into dvd (DVD_ID) values ‘com’||v_count;
v_Count := v_Count + 1;
IF      v_Count >= 100          THEN
EXIT;
END LOOP;
```

Simple LOOP with WHEN Example

```
LOOP
Insert into dvd (DVD_ID) values ‘com’||v_count;
vCount := vCount + 1;
EXIT WHEN     v_Count >= 100 ;
END LOOP;
```

FOR LOOP

FOR i in a .. b LOOP

Statement;

END LOOP;

For LOOP Example

```
BEGIN
FOR SEQ in 10 .. 20 LOOP
Insert into A (PROC) values (SEQ);
END LOOP;
COMMIT;
END;
/
```

*** Executes 11 inserts with seq incrementing by 1 each loop**

TASK!

- Using the PL/SQL program in lab task 6 – Add logic to make the value inserted into the table rows go backwards from 20 to 10
- Be aware that the values in your loop need to move forward in sequence, thus you cannot simply switch the values 10 and 20

6b

Solution!

```
DECLARE
v_count number := 20;
BEGIN
FOR seq in 10 .. 20 LOOP
Insert into A (PROC) values (v_count);
v_count := v_count - 1;
END LOOP;
COMMIT;
END;
/
```

Direct Cursors

```
DECLARE
CURSOR c_movie IS
SELECT type FROM movie;

BEGIN
FOR v_movie in c_movie LOOP
----------------------------------------------
END LOOP;
END;
/
```

What is an direct cursor?

- A MULTIPLE row select query
- Data placed in an area of memory
- Cursor name is a pointer to start of the data
- Data can be processed row by row from beginning to end only
- Columns referenced with the cursor name followed by a dot and the column name

Direct Cursors used in a FOR NEXT loop

```
DECLARE
V_count number := 0;
BEGIN
FOR v_movie in (SELECT type FROM movie) LOOP
IF v_movie.type = 'Drama' then v_count := v_count + 1;
END IF;
END LOOP;
DBMS_OUTPUT.PUT_LINE ('Number of Drama movies: ' ||
v_count);
END;
/
```

Insert the select cursor directly into the in clause

Types of PL/SQL in Oracle

- ANONYMOUS PL/SQL BLOCKS
- NAMED PL/SQL BLOCKS
  - STORED PROCEDURE
  - STORED FUNCTION
  - PACKAGE
  - TRIGGER

Anonymous PL/SQL block

A standalone script holding a block of PL/SQL code which contains a series of PL/SQL commands designed to be parsed and executed at the same time from an SQL utility. The code is not stored in the Oracle DB as an object.

Anonymous PL/SQL block EXAMPLE

```
DECLARE
V_count number := 0;
BEGIN
FOR v_movie in (SELECT TYPE, COUNT(*) CNT FROM
movie GROUP BY TYPE) LOOP
INSERT INTO MOVIE_COUNT (v_movie.type, v_movie.CNT)
END LOOP;
DBMS_OUTPUT.PUT_LINE (‘Movie Counts Inserted …. ');
END;
/
```

*Executing Anonymous PL/SQL block from SQL*Plus*

- **Use get command to retrieve text**
- **SQL> get Movie_block.sql OR**
- **Copy block (CTRL-C) then (CTRL-V)**
- **then use run or / command to execute**
- **SQL> run**
- **SQL> /**

*Executing Anonymous PL/SQL block from SQL*Plus*

- **Save the PL/SQL to a file**
- **SQL> @path\filename**
- **@c:\beg_lab_scripts\8.sql**
- **OR use SQL*Plus Worksheet**

Named PL/SQL blocks

- **Stored as Oracle objects with names**
- **Procedures, Functions, Packages, Triggers**
- **First 3 can be called from other PL/SQL blocks as in “Hello World!” example**

Named PL/SQL block beginning

- **CREATE OR REPLACE PROCEDURE PROC_NAME**
- **SUBPROGRAM AREA**
  - **Identifies variable references passed IN and OUT**

Subprogram Area

- Enclosed by parentheses
- Defines passed parameters both in and out
- Hard-coded or typed data typing
- Separated by commas

Subprogram Area *Example*

```
CREATE OR REPLACE PROCEDURE
EMPLOYEE_SALARY_PRC
(empno IN Number,
deptno IN OUT Varchar2(25),
salary OUT EMPLOYEE.SALARY%type)
```

Subprogram Area Example for Procedures

- **IN - Subprogram needs a variable passed to it, READ ONLY**
- **OUT - Not passed in or read by the subprogram, references an area of memory which can be used to pass data out**
- **IN OUT - Passed variable that references an area of memory that can be altered by the subprogram**

Procedure

A block of PL/SQL code containing a series of statements which can accept and/or return variables by way of referenced memory.

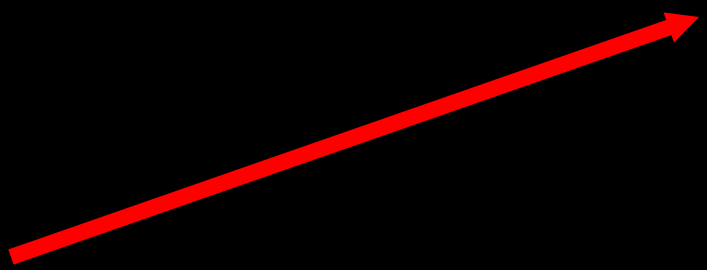
This is done by looking at or changing their value in memory. If changed, that area of memory can be looked at by the calling block.

Procedure Example

```
CREATE OR REPLACE procedure movie_TYPE_prc
(v_type in varchar2)  IS
V_count number := 0;
BEGIN
EXECUTE IMMEDIATE 'TRUNCATE TABLE MOVIE_TEMP';
FOR v_movie in
(SELECT TITLE FROM movie WHERE TYPE = v_type) LOOP
INSERT INTO MOVIE_TEMP (TITLE) VALUES (v_movie.TITLE);
END LOOP;
COMMIT;
DBMS_OUTPUT.PUT_LINE (v_type || ' movies inserted into temp table');
END;
/
SHOW ERRORS
```

Executing a procedure

```
execute movie_TYPE_prc  ('Drama');
```



Pass IN the Type as an argument string

Executing Procedure Examples

From SQL*Plus passing parameters

```
SQL> execute movie_TYPE_prc ('Drama');
```

From SQL*Plus (no parameters)

```
SQL> execute movie_TYPE_prc ();
```

From another calling PL/SQL Block

```
BEGIN
movie_TYPE_prc ('Drama');
END;
```

Function

A block of PL/SQL code containing a series of statements which can accept zero or more variables by memory reference and RETURNING one value to the calling block. An explicit RETURN value statement in the block exits the function and returns the value.

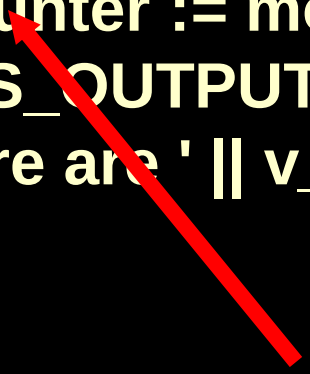
v_Return_Value :=
Function_Name(passed_in_parameter)

Function Example

```
CREATE OR REPLACE function movie_cnt_fnc (v_Type IN
varchar2)
RETURN NUMBER IS
v_Counter number := 0;
BEGIN
SELECT COUNT(*) INTO v_Counter FROM MOVIE
WHERE TYPE = v_Type;
RETURN v_Counter;
EXCEPTION
WHEN NO_DATA_FOUND THEN RAISE;
END;
/
```

Executing a function

```
DECLARE
v_counter number := 0;
V_type varchar2(100);
BEGIN
V_type := 'Horror';
v_Counter := movie_cnt_fnc (v_type);
DBMS_OUTPUT.PUT_LINE
('There are ' || v_counter || ' ' || v_type || ' movies');
END;
/
```



Function returns a numeric value into the variable

Executing Functions

From a calling PL/SQL Block

```
BEGIN
  v_Counter := movie_cnt_fnc (v_type);
END;
```

Executing Functions

From SQL:

```
SQL> select movie_cnt_fnc (v_type)
from dual;
```

Executing Functions

From a sql PREDICATE Clause

```
BEGIN
 IF
 movie_cnt_fnc (v_type) < 30
 THEN
 RAISE_ALERT;
END IF;
END;
```

TASK!

- Using the PL/SQL stored proc in lab task 9 – Create a new stored procedure which being passed the movie type then calls the function created in lab task 10 and passes the movie type on to it for processing
- Rerun the procedure passing different types

Solution!

```
CREATE OR REPLACE procedure movie_TYPE2_prc
(v_type in varchar2)  IS
V_count number := 0;
BEGIN
v_Count := movie_cnt_fnc (v_type);
DBMS_OUTPUT.PUT_LINE
('There are ' || v_count || ' ' || v_type || ' movies');
END;
/
SHOW ERRORS
```

Package

A block of PL/SQL code containing a collection of related procedures and/or functions.

There are two parts to a package: A specification lists the available procedures and functions along with their corresponding parameter declarations. The body contains the actual PL/SQL code.

Package Specification Example

```
CREATE OR REPLACE package movie_pack AS
PROCEDURE movie_count_proc (v_Title IN varchar2);
FUNCTION movie_cnt_fnc (v_Title IN varchar2) return
number;
END;
/
```

Specification must be compiled before the body

Package Body Example

```
CREATE OR REPLACE package body movie_pack AS
    v_Counter := 0;
    PROCEDURE movie_TYPE2_prc IS
    BEGIN
    ----------------------------------------
    END movie_TYPE2_prc;
    FUNCTION movie_cnt_fnc RETURN number IS
    BEGIN
    ----------------------------------------
    END movie_cnt_fnc ;
END;
```

Executing Packaged Procedures

From SQL*Plus passing parameters

```
SQL> execute Moviepack. movie_TYPE2_prc('Horror');
```

From SQL*Plus (no parameters)

```
SQL> execute Moviepack. movie_TYPE2_prc();
```

From another calling PL/SQL Block

```
BEGIN
Moviepack. movie_TYPE2_prc('Horror');
END;
```

Trigger

A block of PL/SQL code containing a series of statements which are attached to an Oracle table. This code is executed whenever a triggering event occurs. This event can be a row inserted, deleted or updated. (individually or any combination)
The code can execute once for each row or can be set to execute just once per event. Update triggers can be set to trigger when individual columns change.

What do you use triggers for?

- To auto populate ID fields from sequences
- To enter date/time and username for update changes to table rows
- To handle foreign key DML changes to multiple tables
- To audit structural or data changes to tables
- Almost anything else!

Trigger Example

```
CREATE OR REPLACE TRIGGER
CREATE_SCRIPT_NAME_TRG
BEFORE INSERT ON DEVTRACKER
FOR EACH ROW
begin
SELECT LPAD(SCRIPT_SEQ.NEXTVAL,5,'0')||'-01-
DT'||:new.DT_NUM
    INTO   :new.SCRIPT
    FROM   dual;
:new.SCRIPT_DATE := SYSDATE;
end;
/
```

Trigger Example with 11g change direct assignment from sequences

```
CREATE OR REPLACE TRIGGER
CREATE_SCRIPT_NAME_TRG
BEFORE INSERT ON DEVTRACKER
FOR EACH ROW
begin
:new.SCRIPT := LPAD(SCRIPT_SEQ.NEXTVAL,5,'0')||'-01-
DT'||:new.DT_NUM;
:new.SCRIPT_DATE := SYSDATE;
end;
/
```

Conventions for calling named PL/SQL blocks

- Simple - just the procedure/function name proc_name
- Qualified - package_name.proc_name
- Remote - name + @remote_loc
- Qualified and Remote - package_name.proc_name@remote.loc

Final Thoughts

- PL/SQL is an extraordinary excellent way to transfer code functionality to the database
- Improve performance time for data transfers
- Simplified Maintenance
- Easy to Code
- Constantly improving

Final Thoughts

- Oracle keeps adding new functionality to it’s packages and more packages
- Break complicated logic into simpler blocks of called code
- Automate simple tasks like statistic creation
- Handy functions make life simpler

"I see PL/SQL in your Future!"

Contact Info

- sam.weber@attask.com
- Contact me if you have follow-up questions
- Check Out www.attask.com on the internet, it’s a great Project and Portfolio management software for project managers, developers and executives

Questions?