IT 4153 Advanced Database

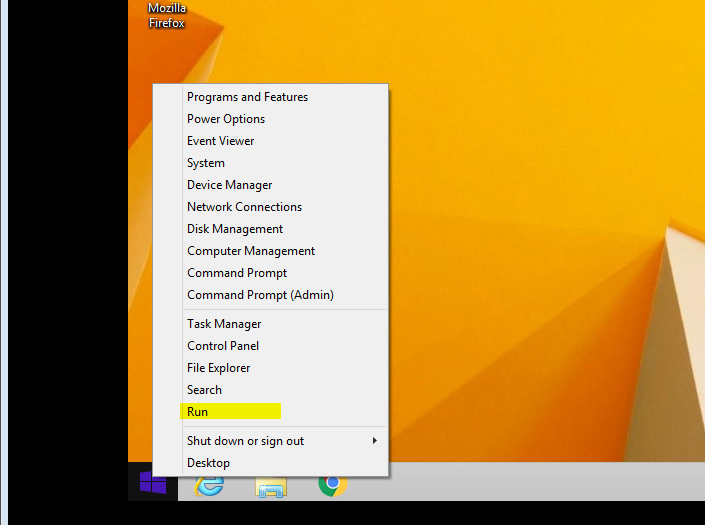
Module 3 Advanced SQL Procedures & Functions

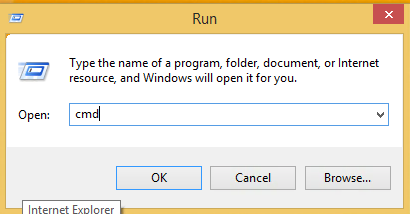
|  |
| --- |
| **Introduction and Module Summary** |
| In this module, you will learn how to use IF, CASE, COMMIT, and LOOPs. |
| **Objectives and Outcomes** |
| This module directly supports **highlighted** course outcome(s)  Students who complete this course successfully will be able to   1. Describe current and emerging database models and technologies; 2. **Develop functions and procedures for data manipulation and database access auditing;** 3. Describe database monitoring and performance tuning; 4. Describe database security and administration issues, including backup and recovery; 5. Explain the concepts of data warehousing and data mining   **Module outcomes and activities:**   |  |  |  |  | | --- | --- | --- | --- | | After completing this module, students will be able: | To develop Oracle PL/SQL code that requires use of PL/SQL control structures | To test Oracle PL/SQL code that requires use of PL/SQL control structures | To debug Oracle PL/SQL code that requires use of PL/SQL control structures | | Readings | introduced | introduced | introduced | | Practice exercises | reinforced | reinforced | reinforced | | Discussion | reinforced |  |  | | Lab | mastered | mastered | reinforced | |
| **Assigned Reading** |
| 1. Using PL/SQL Control Structures <http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/controlstructures.htm#BABDAEGB> |
| **Optional Reading** |
| 1. Oracle control structures <http://psoug.org/reference/control_struct.html> |
| **Assessments and Assignments** |
| 1. Lab (10 points) 2. Discussion (5 points) |
| **Topics** |
| Open the navigation pane |
| **Module Checklist** |
| This is the suggested order of the completion of this module.  Save a copy of this file on your computer and make notes in this document while you are completing your assignments. Use the table below to keep track of your progress.   |  |  | | --- | --- | | **Activity** | **Completion** | | Read this module and assigned materials (1 hour) | NO | | Complete all exercises from the module (2 hours) | NO | | Prepare for discussion and post your example (1 hour) | NO | | Complete the lab (2 hours) | NO | | Complete a feedback section at the end of the module | NO | | Read feedback provided for your discussion and lab. | NO | |
|  |

# COMMIT

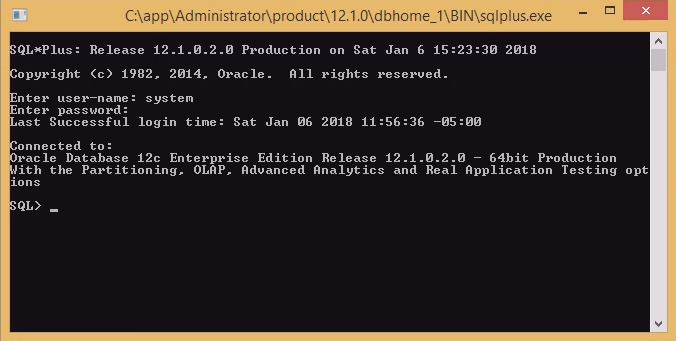
The COMMIT statement makes permanent any changes made to the database during the current transaction. A commit also makes the changes visible to other users.

Suppose as a system user I will create a table using sql\*Plus and insert one row. To start sql\*Plus, right-click Windows button and choose run, then type cmd and press OK.





In the new window type ***sqlpplus*** and press enter.



Use system as username and the usual password (see http://ksuweb.kennesaw.edu/~speltsve/files/sql.php).

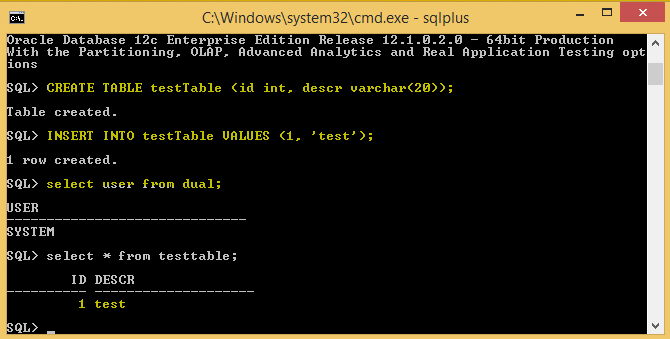
CREATE TABLE testTable (id int, descr varchar2(20));

INSERT INTO testTable VALUES (1, 'test');

Let's check that table was created:

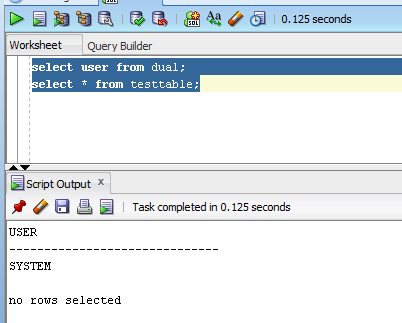
SELECT USER FROM DUAL;

SELECT \* FROM TESTTABLE;



The result is one row.

Open SQL developer, login as system and check how many rows the table has.

.

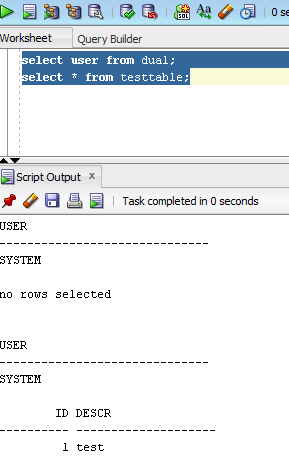
There are no rows returned. The table exists, but data is not there.

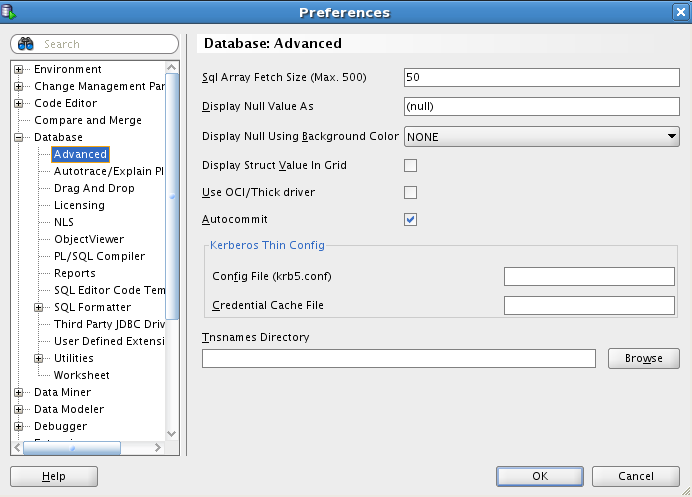
Oracle Database issues an implicit COMMIT before and after any data definition language (DDL) statement.

Why insert statement was not committed? DML statements are not implicitly committed. As soon as commit issued or session is closed, other sessions will see the inserted row

COMMIT;

As soon as commit is issued in SQL\*Plus window, one row is visible in SQL Developer session



Some database clients have autocommit options e.g. SQL Developer (Tools > Preferences > Database > Advanced)

Oracle guaranties a read consistent view of the data, so all inserted and updated data is held in memory and is available only to a current user. COMMIT or ROLLBACK releases this lock. Transactions can be controlled by using SAVEPOINT statement

CREATE TABLE testTable2 (id int, descr varchar2(20));

SAVEPOINT NoRecords; --point we want to remember

INSERT INTO testTable2 VALUES (1, 'test');

SAVEPOINT OneRecord; --point we want to remember

INSERT INTO testTable2 VALUES (2, 'second');

SELECT \* FROM testTable2; --select returns 2 rows

ROLLBACK TO OneRecord;

SELECT \* FROM testTable2; --select returns 1 row

ROLLBACK TO NoRecords; --select returns 0 rows

SELECT \* FROM testTable2;

The result:

table TESTTABLE2 created.

SAVEPOINT NoRecords

1 rows inserted.

SAVEPOINT OneRecord

1 rows inserted.

ID DESCR

-------------------------------------- --------------------

1 test

2 second

rollback complete.

ID DESCR

-------------------------------------- --------------------

1 test

rollback complete.

no rows selected

# Testing Conditions (IF and CASE Statements)

## IF-ELSE

CREATE OR REPLACE FUNCTION MAX\_VALUE(x int, y int)

RETURN int AS

BEGIN

IF x > y THEN

RETURN x;

ELSE

RETURN y;

END IF;

END;

/

SET SERVEROUTPUT ON;

SELECT MAX\_VALUE(7,9) FROM DUAL; --returns 9

SELECT MAX\_VALUE(45,3) FROM DUAL; --returns 45

Another example of IF-THEN-ELSE, but using unnamed block and some useful date operations.

We want to write a code that prompts user to input a date and then tells if it is fall, spring, summer or winter day.

SET SERVEROUTPUT ON;

DECLARE

user\_date DATE;

user\_day INT;

user\_month INT;

BEGIN

user\_date := TO\_DATE('&please\_input\_date\_to\_check', 'DD-MON-YYYY');

-- ampersand means that user will be prompted for input

-- DD-MON-YYYY e.g. 06-NOV-2018

user\_day := RTRIM(TO\_CHAR(user\_date,'DD'));

user\_month := RTRIM(TO\_CHAR(user\_date,'MM'));

IF ( (user\_month = 1) OR (user\_month = 2)) THEN

Dbms\_output.put\_line('The season is Winter');

ELSIF( (user\_month = 4) OR (user\_month = 5)) THEN

Dbms\_output.put\_line('The season is Spring');

ELSIF( (user\_month = 7) OR (user\_month = 8)) THEN

Dbms\_output.put\_line('The season is Summer');

ELSIF( (user\_month = 10)OR (user\_month = 11)) THEN

Dbms\_output.put\_line('The season is Fall');

ELSIF( (user\_month = 3) AND (user\_day <= 19 )) THEN

Dbms\_output.put\_line('The season is Winter');

ELSIF( (user\_month = 3) AND (user\_day >= 20 )) THEN

Dbms\_output.put\_line('The season is Spring');

ELSIF( (user\_month = 6) AND (user\_day <= 20 )) THEN

Dbms\_output.put\_line('The season is Spring');

ELSIF( (user\_month = 6) AND (user\_day >= 21 )) THEN

Dbms\_output.put\_line('The season is Summer');

ELSIF( (user\_month = 9) AND (user\_day <= 20 )) THEN

Dbms\_output.put\_line('The season is Summer');

ELSIF( (user\_month = 9) AND (user\_day >= 21 )) THEN

Dbms\_output.put\_line('The season is Fall');

ELSIF( (user\_month = 12) AND (user\_day <= 21 )) THEN

Dbms\_output.put\_line('The season is Fall');

ELSIF( (user\_month = 12) AND (user\_day >= 22 )) THEN

Dbms\_output.put\_line('The season is Winter');

END IF;

END;

/

In Oracle/PLSQL ***TO\_CHAR*** function converts a number or date to a string.

*TO\_CHAR( value, [ format\_mask] )* [ ] means optional

value - a number or a date

format\_mask - format that will be used to convert value to a string

Examples:

TO\_CHAR(45350.3, '$99,999.00') would return $45,350.30

TO\_CHAR(45350.3, '$9,999.00') would return ##########

format mask is not long enough

TO\_CHAR(453, '0000999') would return 0000453 (7 digits)

TO\_CHAR(453, '00099') would return 00453 (5 digits)

TO\_CHAR(453, '009999') would return 000453 (6 digits)

TO\_CHAR(TO\_DATE('11-NOV-2011'),'DAY') would return Friday

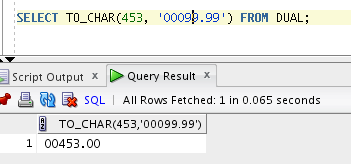
TO\_CHAR(TO\_DATE('11-NOV-2011'),'DD') would return 11 (day of the month)

TO\_CHAR(TO\_DATE('11-NOV-2011'),'MM') would return 11 (November)

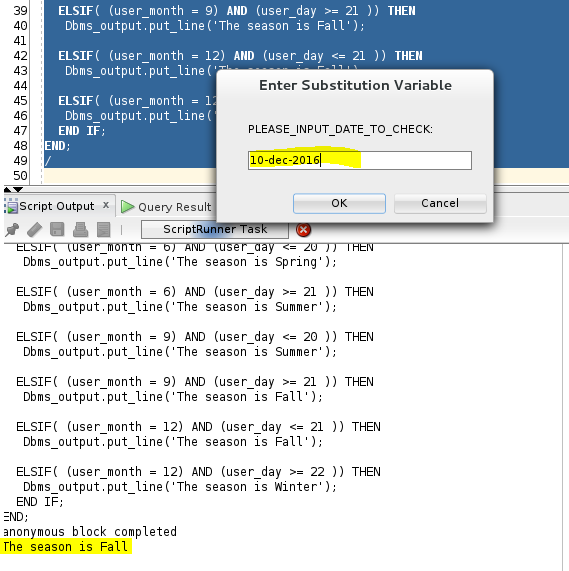
More examples available at <http://www.oradev.com/oracle_number_format.jsp> and <http://www.oradev.com/oracle_date_format.jsp>

To test examples use:

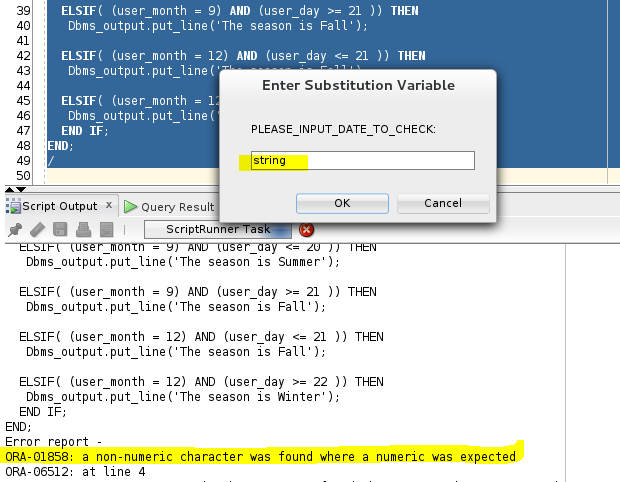
SELECT TO\_CHAR(453, '00099.99') FROM DUAL;



The result of the previous code is



If input string cannot be converted to the date format, then an error message " ORA-01858: a non-numeric character was found where a numeric was expected" will be displayed.

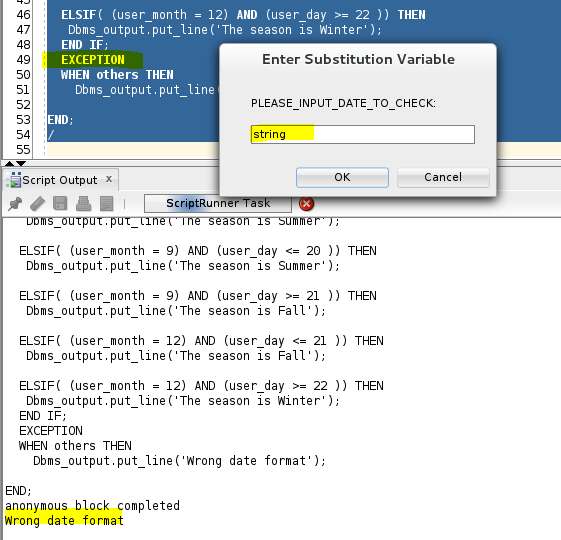


To avoid this error, we have to add code to handle the exception. Add the following code right before the end of the block

EXCEPTION

WHEN others THEN

Dbms\_output.put\_line('Wrong date format');



Read more about date and time Oracle data types at <http://infolab.stanford.edu/~ullman/fcdb/oracle/or-time.html>

We will use OE schema (Order Entry) tables PRODUCT\_DESCRIPTION and PRODUCT\_INFORMATION. Description of OE schema can be found at <http://download.oracle.com/docs/cd/B28359_01/server.111/b28328.pdf> (p. 20 & p. 24)

Create a table, insert data in the table and then read that data using PL/SQL stored procedure

Create table *ProductTable*

|  |  |  |  |
| --- | --- | --- | --- |
| **ProductID** | **ProductName** | **ListPrice** | **Category** |
| int, not null | Not null, length 50 | $ | int, not null |

Add values to table ProductTable

|  |  |  |  |
| --- | --- | --- | --- |
| **ProductID** | **ProductName** | **ListPrice** | **Category** |
| 299 | Chest | $99.99 | 10 |
| 300 | Wave Cruiser | $49.99 | 11 |
| 301 | Megaland Play Tent | $59.99 | 11 |
| 302 | Wind-Up Water Swimmers tent | $2.00 | 11 |
| 303 | Garmin Pocket or Vehicle GPS Navigator | $609.99 | 12 |

CREATE TABLE ProductTable(

ProductID INTEGER NOT NULL primary key,

ProductName VARCHAR(50) NOT NULL,

ListPrice NUMBER(10,2),

Category INTEGER NOT NULL

);

/

INSERT INTO ProductTable VALUES(299,'Chest',99.99,10);

INSERT INTO ProductTable VALUES(300,'Wave Cruiser',49.99,11);

INSERT INTO ProductTable VALUES(301,'Megaland Play Tent',59.99,11);

INSERT INTO ProductTable VALUES(302,'Wind-Up Water Swimmers tent',2.00,11);

INSERT INTO ProductTable VALUES(303,'Garmin Pocket or Vehicle GPS Navigator',609.99,12);

1. To find all products from ProductTable table with a name that contains "Tent".
   1. Run

select \* from ProductTable where ProductName like '%Tent%';

The result: one product with id 301

* 1. Run

select \* from ProductTable where lower(ProductName) like '%tent%';

The result: 2 products

301 and 302

1. Write PL/SQL function that takes a string as input, then checks if we have products that matches the search parameter

create or replace

function search\_by\_description(keyword VARCHAR2)

RETURN VARCHAR2

AS

num\_products INT;

BEGIN

SELECT COUNT(\*) INTO num\_products

FROM ProductTable

WHERE LOWER(ProductName)

LIKE '%' || keyword || '%';

IF num\_products > 0 THEN

RETURN keyword || ' found ' || num\_products;

ELSE

RETURN keyword || ' not found';

END IF;

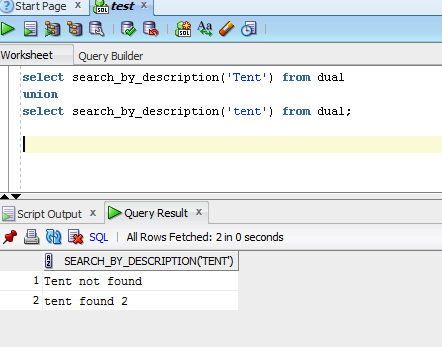
END;

/

select search\_by\_description('Tent') from dual

union

select search\_by\_description('tent') from dual;



I used UNION to show results of two calls to the function. The second select returned two products and the first one did not return any.

**Question:** Why in #2 above we had 2 products matching "Tent" and now non were found? How can it be corrected?

You have to modify the select statement to use LOWER(keyword) instead of just keyword.

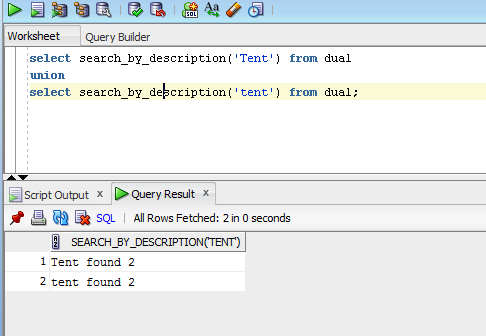
SELECT COUNT(\*) INTO num\_products

FROM ProductTable

WHERE LOWER(ProductName)

LIKE '%' || keyword || '%';

The result will be two rows for both selects:



## CASE

Now we will rewrite this function to use CASE statement

CASE

WHEN num\_products > 1 THEN RETURN keyword || ' found more than one';

WHEN num\_products > 0 THEN RETURN keyword || ' less than two found';

ELSE RETURN keyword || ' not found';

END CASE;

If two or more products are found, the message will be 'found more than one' any other positive number will generate 'less than two found'

The result is:

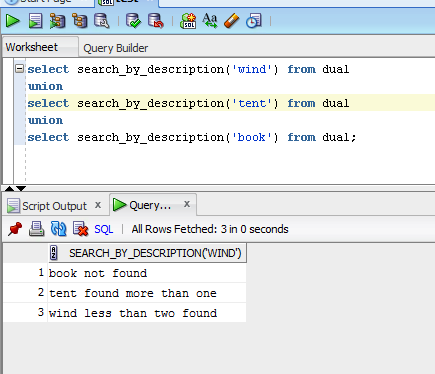
select search\_by\_description('wind') from dual

union

select search\_by\_description('tent') from dual

union

select search\_by\_description('book') from dual;



## IF-ELSIF

The same code can be written using ELSIF

IF num\_products > 1 THEN RETURN keyword || ' found more than one';

ELSIF num\_products > 0 THEN RETURN keyword || ' less than two found';

ELSE RETURN keyword || ' not found';

END IF;

The CASE statement is like a series of IF statements, if no condition is found to be true, then the case statement will return the value in the ELSE clause.

SET SERVEROUTPUT ON;

DECLARE

grade CHAR(1);

BEGIN

grade := 'B';

CASE grade

WHEN 'A' THEN DBMS\_OUTPUT.PUT\_LINE('Excellent');

WHEN 'B' THEN DBMS\_OUTPUT.PUT\_LINE('Very Good');

WHEN 'C' THEN DBMS\_OUTPUT.PUT\_LINE('Good');

WHEN 'D' THEN DBMS\_OUTPUT.PUT\_LINE('Fair');

WHEN 'F' THEN DBMS\_OUTPUT.PUT\_LINE('Poor');

ELSE DBMS\_OUTPUT.PUT\_LINE('No such grade');

END CASE;

END;

/

The result: Very Good

# Controlling Loop Iterations (LOOP, EXIT, and CONTINUE Statements)

## Basic loop

SET SERVEROUTPUT ON;

DECLARE

i INT;

BEGIN

i := 0;

LOOP

IF i <= 10 THEN

i := i + 1;

dbms\_output.put\_line(i);

ELSE

EXIT;

END IF;

END LOOP;

END;

/

This code will display numbers from 1 to 11 (one per line). To display numbers on one line use

dbms\_output.put(i || ', ');

The output is buffered and to see the output you need to add end-of-line to the buffer either using put\_line or new\_line.

## FOR Loop

SET SERVEROUTPUT ON;

DECLARE

i INT;

BEGIN

FOR i in 0..10 LOOP

dbms\_output.put(i || ', ');

END LOOP;

dbms\_output.new\_line();

END;

/

This code will display numbers from 0 to 10 on one line.

## WHILE Loop

SET SERVEROUTPUT ON;

DECLARE

i INT;

BEGIN

i := 0;

WHILE i <= 10 LOOP

dbms\_output.put(i || ', ');

i := i + 1;

END LOOP;

dbms\_output.new\_line();

END;

/

This code will display numbers from 0 to 10 on one line.

## Using EXIT and CONTINUE

CONTINUE stops the current iteration of the loop and starts the next iteration of the loop. You have to be very careful when you use CONTINUE with WHILE loops, it can cause an infinite loop.

SET SERVEROUTPUT ON;

DECLARE

i INT;

BEGIN

FOR i in 0..10 LOOP

IF MOD(i, 2) = 0 THEN --if the number is even

CONTINUE;

END IF;

--this statement will be reached only if the number is odd

dbms\_output.put(i || ', ');

END LOOP;

dbms\_output.new\_line();

END;

/

This code will display numbers 1, 3, 5, 7, 9,

EXIT statement is used when condition of exit is not known before the loop starts. Suppose we want to find factorials up to 1000000

SET SERVEROUTPUT ON;

DECLARE

i NUMBER := 1;

factorial NUMBER := 1;

BEGIN

WHILE TRUE LOOP

factorial := factorial \* i;

IF factorial > 1000000 THEN

EXIT;

END IF;

DBMS\_OUTPUT.PUT\_LINE('Factorial of ' || i || ' is ' || factorial);

i := i + 1;

END LOOP;

END;

/

The result will be

anonymous block completed

Factorial of 1 is 1

Factorial of 2 is 2

Factorial of 3 is 6

Factorial of 4 is 24

Factorial of 5 is 120

Factorial of 6 is 720

Factorial of 7 is 5040

Factorial of 8 is 40320

Factorial of 9 is 362880

Factorial of 10 will be calculated, but not displayed because it is equal to 3628800.

# Lab

1. Post a question at the Help with Labs section of the discussion board and answer two questions posted on the board. If you do not have questions, you can post a link to a good recourse to complement the module reading material (article, example or youtube video). Two replies are still required. (2 points)

2. Create a table UserPermissions (provide create and insert statements code) (5 points for the correct code of the procedure and 3 points for testing your procedure; one point per test case). Do not modify table or column names.

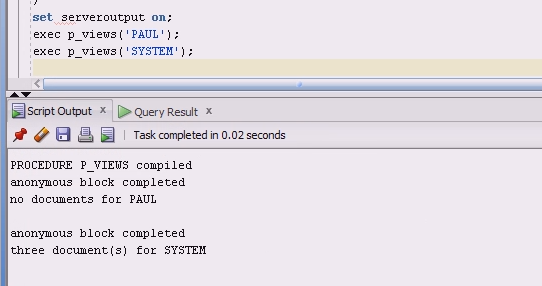
|  |  |
| --- | --- |
| **Document** | **UserName** |
| Policy | SYSTEM |
| Menu | JDOW |
| W2 | USAM |
| Permissions | SYSTEM |
| W2 | JDOW |
| Form 1040 | USAM |
| Policy | JDOW |
| W2 | SYSTEM |

Write a PL/SQL stored procedure that takes ***username*** as input and returns number of documents that user has permissions to view. If ***username*** is not in the table, your procedure should return: "no documents for USERNAME".

The number of documents has to be displayed as a word instead of using digits. To convert number to words read <http://viralpatel.net/blogs/convert-number-into-words-oracle-sql-query/>

Include code that can be copied and ran, and screenshots of the results. Do not forget to test "user not found" case.

Sample output:



**Feedback:**

Difficulty (-2 - too easy ... 0 - just right ... 2 - too hard)

Interest level (-2 - low interest ... 0 - just right ... 2 - high interest)

Time to complete (min)

Make a suggestion to improve

What to submit:

One report file that contains PL/SQL code (not screenshots of code) and screenshots of the results to prove functionality.

File name and size should conform to submission standards <http://ksuweb.kennesaw.edu/~speltsve/files/style_and_submission_guide_d2l.pdf>