```
DECLARE
        TYPE t_ManagerTable IS TABLE OF employeez.mgr%TYPE
          INDEX BY BINARY_INTEGER;

        v_ManagerIdTable t_ManagerTable;

        CURSOR c_Employee1 IS
          SELECT empno, sal, mgr FROM employeez
          ORDER BY mgr
          FOR UPDATE OF sal;
        v_EmployeeId employeez.empno%TYPE;
        v_ManagerId employeez.mgr%TYPE;
        v_Salary employeez.sal%TYPE;
        v_FoundFlag BOOLEAN;
        v_LoopCnt BINARY_INTEGER;

BEGIN
        -- Build a table of manager ID's
        OPEN c_Employee1;
        FETCH c_Employee1 into v_EmployeeId, v_Salary, v_ManagerId;
        WHILE c_Employee1%FOUND LOOP
          IF v_ManagerIdTable.COUNT = 0

           OR v_ManagerId !=
             v_ManagerIdTable(v_ManagerIdTable.COUNT) THEN
             v_ManagerIdTable(v_ManagerIdTable.COUNT + 1) :=
             v_ManagerId;
          END IF;
          FETCH c_Employee1 into v_EmployeeId, v_Salary, v_ManagerId;
        END LOOP;
        CLOSE c_Employee1;

        -- Look in manager ID table to determine if employee is a manager
        -- If employee is a manager increase pay by 10% else by 2%

       OPEN c_Employee1;
```

This is a version of #4 in the last set of cursor problems that uses the while instead of the for.

```
      END IF;
      FETCH c_Employee1 into v_EmployeeId, v_Salary, v_ManagerId;
    END LOOP;
    CLOSE c_Employee1;

    -- Look in manager ID table to determine if employee is a manager
    -- If employee is a manager increase pay by 10% else by 2%

   OPEN c_Employee1;
   FETCH c_Employee1 into v_EmployeeId, v_Salary, v_ManagerId;
   WHILE c_Employee1%FOUND LOOP
      v_FoundFlag := FALSE;
      v_LoopCnt  := 1;
      WHILE (v_LoopCnt <= v_ManagerIdTable.COUNT) AND
       (v_FoundFlag = FALSE) LOOP
        IF v_EmployeeId = v_ManagerIdTable(v_LoopCnt) THEN
          v_FoundFlag := TRUE;
        END IF;
        v_LoopCnt := v_LoopCnt + 1;
      END LOOP;
      IF v_FoundFlag = TRUE THEN
        v_Salary := 1.1 * v_Salary;
      ELSE
        v_Salary := 1.02 * v_Salary;
      END IF;
      UPDATE employeez
        SET sal = v_Salary
        WHERE CURRENT OF c_Employee1;
      FETCH c_Employee1 into v_EmployeeId, v_Salary, v_ManagerId;
    END LOOP;

    COMMIT;
END;
/
SET SERVEROUTPUT OFF
```

Title: Nov 14-9:46 AM (2 of 20)

File Edit View History Bookmarks Tools Help

Introduction to procedures

www.pgrocer.net/Cis50/procedures.html

The structure of a named block such as a procedure or function as opposed to the anonymous blocks that we have seen so far in this course, is shown below:

Procedures

```
Header (used for named block/modules only)
IS
   Declaration section
BEGIN
   Processing/execution section
EXCEPTION
   Exception section
END;
```

For syntax and more in-depth descriptions, please check your text or the locations on the Web that have been suggested.

I am going to create a procedure to add donations to the donation table. Since I don't want to mess up the actual table, I have created a copy as shown below.

SQL CODE:

```
SQL> CREATE TABLE new_donation AS
  2  SELECT * FROM donation;

Table created.

SQL> SELECT * FROM new_donation;

IDNO  DRI CONTDATE    CONTAMT
----- --- --------- ---------
11111 100 07-JAN-99        25
12121 200 23-FEB-99        40
23456 100 03-MAR-99        20
33333 300 10-MAR-99        10
22222 100 14-MAR-99        10
12121 100 04-JUN-99        50
11111 200 12-JUN-99        35
23456 300 14-JUN-99        10

8 rows selected.
```

The code below creates a procedure with the name AddDonation. Notice that there is no DECLARE in this block, the IS or AS serves the purpose. This procedure sets up variables that will receive data when the procedure is called. The insert uses these variables to add a record to the new_donations table. After I have written the code, I did the @ adddonation which came back with the message, procedure created.

SQL CODE:

```
SQL> edit adddonation
```

PL/SQL CODE:

```
CREATE OR REPLACE PROCEDURE AddDonation
  (p_idno        new_donation.idno%TYPE,
   p_driveno     new_donation.driveno%TYPE,
   p_contdate    new_donation.contdate%TYPE,
   p_contamt     new_donation.contamt%TYPE) AS
BEGIN
   INSERT INTO new_donation(idno, driveno, contdate, contamt)
```

Type here to search

9:49 AM 11/14/2019

```
SQL> edit adddonation
```

PL/SQL CODE:

```
CREATE OR REPLACE PROCEDURE AddDonation
  (p_idno        new_donation.idno%TYPE,
   p_driveno     new_donation.driveno%TYPE,
   p_contdate    new_donation.contdate%TYPE,
   p_contamt     new_donation.contamt%TYPE) AS
BEGIN
   INSERT INTO new_donation(idno, driveno, contdate, contamt)
      VALUES(p_idno, p_driveno, p_contdate, p_contamt);
END AddDonation;
/
```

SQL CODE:

```
SQL> @ adddonation
```

Remember that once the procedure is written you have to use @ and create it.

```
Procedure created.
```

Next a wrote a block of PL/SQL code that would call the procedure that I have just written. I took in user input for the record that was being created. Notice again the quotes around non-numeric fields. I then decided to use an IF and call the procedure only if the v_contamt that the user entered was greater than 20. The call itself uses the procedure name and passes the variables containing data for the new record to the procedure. In the procedure above, I started out by giving the procedure the name AddDonation and then used parenthesis to list the parameters that would be received. Compare the parameters that would be received in AddDonation to the parameters that are sent when the call is given. Notice that they match. Again notice that in the procedure after the parameters in parenthesis the IS or AS is used followed by the BEGIN. The parameters in the procedure are called formal parameters and the parameters in the calling block are called actual parameters. The actual parameters are passed to the formal parameters and they can also receive information if information is passed back.

SQL CODE:

```
SQL> edit call_adddonation
```

PL/SQL CODE:

This is the code that is talking in data, calling the procedure and passing it the data.

```
SET VERIFY OFF
DECLARE
  v_idno       new_donation.idno%TYPE :='&input_idno';
  v_driveno    new_donation.driveno%TYPE :='&input_driveno';
  v_contdate   new_donation.contdate%TYPE :='&input_contdate';
  v_contamt    new_donation.contamt%TYPE :=&input_contamt;
BEGIN
  IF v_contamt > 20 THEN
     AddDonation (v_idno, v_driveno, v_contdate, v_contamt);
  END IF;
END;
/
SET VERIFY ON
```

SQL CODE:

```
SQL> @ call_adddonation
Enter value for input_idno: 12121
Enter value for input_driveno: 300
```

Title: Nov 14-9:54 AM (4 of 20)

File Edit View History Bookmarks Tools Help

Introduction to procedures

www.pgrocer.net/Cis50/procedures.html

PL/SQL CODE:

```
SET VERIFY OFF
DECLARE
  v_idno       new_donation.idno%TYPE :='&input_idno';
  v_driveno    new_donation.driveno%TYPE :='&input_driveno';
  v_contdate   new_donation.contdate%TYPE :='&input_contdate';
  v_contamt    new_donation.contamt%TYPE :=&input_contamt;
BEGIN
  IF v_contamt > 20 THEN
     AddDonation (v_idno, v_driveno, v_contdate, v_contamt);
  END IF;
END;
/
SET VERIFY ON
```

SQL CODE:

```
SQL> @ call_adddonation
Enter value for input_idno: 12121
Enter value for input_driveno: 300
Enter value for input_contdate: 10-JUN-99
Enter value for input_contamt: 75

PL/SQL procedure successfully completed.

Input truncated to 5 characters
SQL> SELECT * FROM new_donation;

IDNO  DRI CONTDATE    CONTAMT
----- --- --------- ---------
11111 100 07-JAN-99        25
12121 200 23-FEB-99        40
23456 100 03-MAR-99        20
33333 300 10-MAR-99        10
22222 100 14-MAR-99        10
12121 100 04-JUN-99        50
11111 200 12-JUN-99        35
23456 300 14-JUN-99        10
12121 300 10-JUN-99        75

9 rows selected.
```

Going a little further, we find that there are three modes for formal parameters: IN, OUT or IN OUT. IN sends the parameters from the calling program to the procedure. OUT sends the parameters from the procedure back to the calling program. IN OUT passes to and receives from the procedure. The default mode is IN. Since no mode was s pecified in the example above, the default was IN. To test this, I created the following table:

SQL CODE:

```
SQL> CREATE TABLE inout_table
  2  (col_in NUMBER, col_out NUMBER, col_inout NUMBER);

Table created.

SQL> DESC inout_table
 Name                            Null?    Type
 ------------------------------- -------- ----
```

Type here to search

9:55 AM
11/14/2019

File Edit View History Bookmarks Tools Help

Introduction to procedures

www.pgrocer.net/Cis50/procedures.html

```
END;
/
```

SQL CODE:

```
SQL> @ proc_calc

Procedure created.

SQL> edit calc_proc_calc
```

PL/SQL CODE:

```
DECLARE
   v_in    NUMBER :=&input_in;
   v_out   NUMBER :=&input_out;
   v_inout NUMBER :=&input_inout;
BEGIN
   proc_calc(v_in, v_out, v_inout);
END;
/
```

SQL CODE:

```
SQL> @ calc_proc_calc

Input truncated to 1 characters
Enter value for input_in: 15
old   2:    v_in    NUMBER :=&input_in;
new   2:    v_in    NUMBER :=15;
Enter value for input_out: 72
old   3:    v_out   NUMBER :=&input_out;
new   3:    v_out   NUMBER :=72;
Enter value for input_inout: 99
old   4:    v_inout NUMBER :=&input_inout;
new   4:    v_inout NUMBER :=99;

PL/SQL procedure successfully completed.

SQL> SELECT * FROM inout_table;


   COL_IN    COL_OUT COL_INOUT
--------- --------- ---------
       15        30        45
```

I then altered the procedure (proc_calc) to add the input that was received into p_inout and the variable that was received in p_in and store the result in p_inout. I inputed 23, 33, 43 and the output acted as expected and added the 43 +23 to get 66.

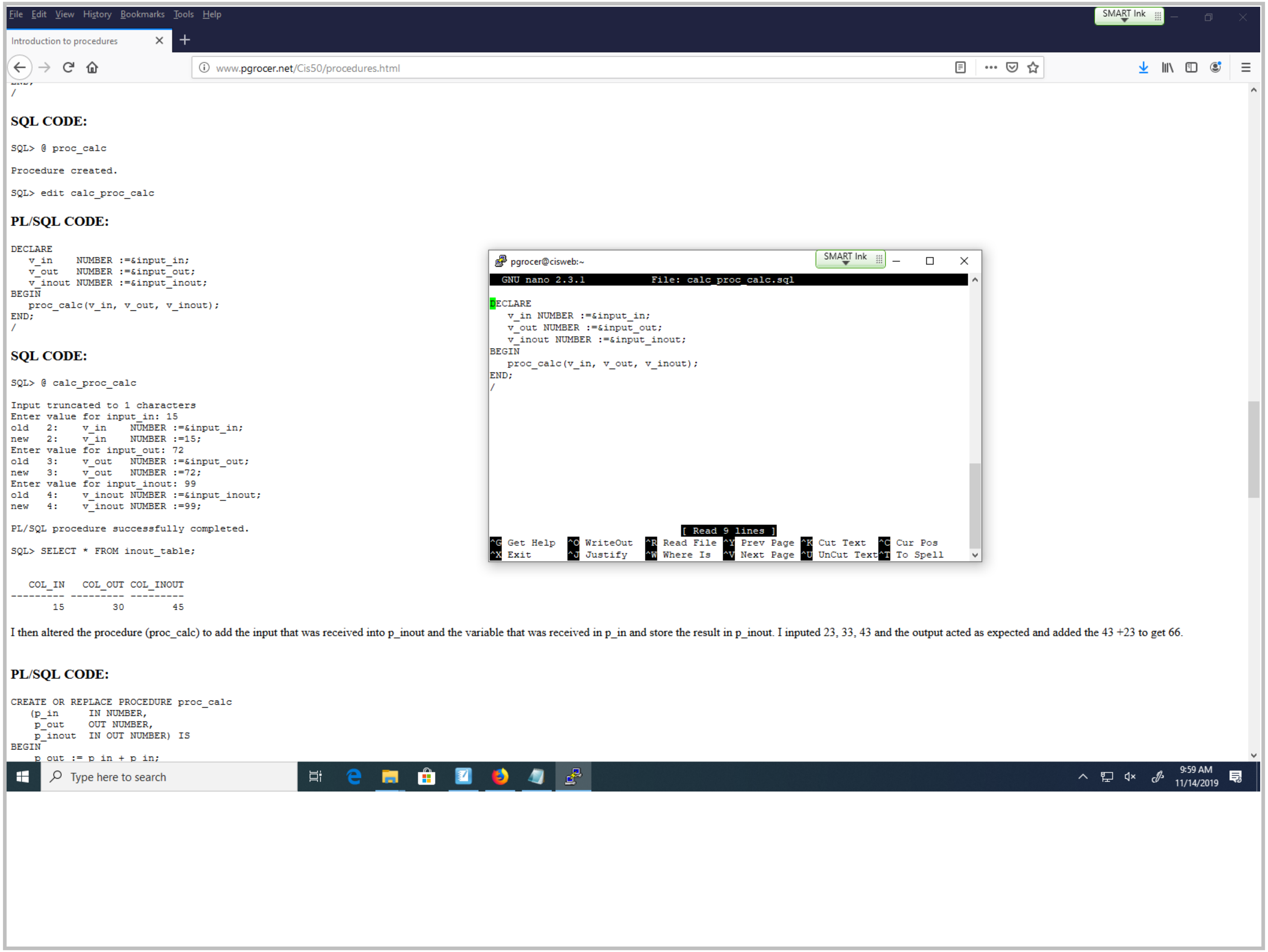
PL/SQL CODE:

```
CREATE OR REPLACE PROCEDURE proc_calc
   (p_in      IN NUMBER,
    p_out     OUT NUMBER,
    p_inout   IN OUT NUMBER) IS
BEGIN
    p_out := p_in + p_in;
```

Type here to search

9:58 AM
11/14/2019

Title: Nov 14-9:58 AM (6 of 20)



Introduction to procedures

www.pgrocer.net/Cis50/procedures.html

```
/
```

SQL CODE:

```
SQL> @ proc_calc

Procedure created.

SQL> edit calc_proc_calc
```

PL/SQL CODE:

```
DECLARE
   v_in    NUMBER :=&input_in;
   v_out   NUMBER :=&input_out;
   v_inout NUMBER :=&input_inout;
BEGIN
   proc_calc(v_in, v_out, v_inout);
END;
/
```

SQL CODE:

```
SQL> @ calc_proc_calc

Input truncated to 1 characters
Enter value for input_in: 15
old   2:    v_in      NUMBER :=&input_in;
new   2:    v_in      NUMBER :=15;
Enter value for input_out: 72
old   3:    v_out     NUMBER :=&input_out;
new   3:    v_out     NUMBER :=72;
Enter value for input_inout: 99
old   4:    v_inout NUMBER :=&input_inout;
new   4:    v_inout NUMBER :=99;

PL/SQL procedure successfully completed.

SQL> SELECT * FROM inout_table;


   COL_IN    COL_OUT COL_INOUT
--------- --------- ---------
       15        30        45
```

I then altered the procedure (proc_calc) to add the input that was received into p_inout and the variable that was received in p_in and store the result in p_inout. I inputed 23, 33, 43 and the output acted as expected and added the 43 +23 to get 66.

PL/SQL CODE:

```
CREATE OR REPLACE PROCEDURE proc_calc
   (p_in     IN NUMBER,
    p_out    OUT NUMBER,
    p_inout  IN OUT NUMBER) IS
BEGIN
    p_out := p_in + p_in;
```


Title: Nov 14-9:59 AM (7 of 20)

```
COL_INOUT                                 NUMBER
```

In the example below, there is a procedure and a calling PL/SQL block. The calling block takes in three numbers and passes then to the procedure. In the sample run, I inputed 15, 72 and 99. These three numbers were passed to the proc_calc procedure. The input number is passed on a read only basis so I did no processing with the p_in. The output number is passed on a write only basis and so the 72 is ignored and p_out is calculated within the program as 15 + 15. The input/output number comes in as 99 but within the program I add 15 + 15 + 15 so this is the number that appears on the table through the insert.

SQL CODE:

```
SQL> edit proc_calc
```

PL/SQL CODE:

```
CREATE OR REPLACE PROCEDURE proc_calc
   (p_in      IN NUMBER,
    p_out     OUT NUMBER,
    p_inout   IN OUT NUMBER) IS
BEGIN
    p_out := p_in + p_in;
    p_inout := p_in + p_in + p_in;
    INSERT INTO inout_table (col_in, col_out, col_inout)
      VALUES (p_in, p_out, p_inout);
END;
/
```

SQL CODE:

```
SQL> @ proc_calc

Procedure created.

SQL> edit calc_proc_calc
```

PL/SQL CODE:

```
DECLARE
   v_in     NUMBER :=&input_in;
   v_out    NUMBER :=&input_out;
   v_inout NUMBER :=&input_inout;
BEGIN
   proc_calc(v_in, v_out, v_inout);
END;
/
```

SQL CODE:

```
SQL> @ calc_proc_calc

Input truncated to 1 characters
Enter value for input_in: 15
old   2:    v_in     NUMBER :=&input_in;
new   2:    v_in     NUMBER :=15;
Enter value for input_out: 72
old   3:    v_out    NUMBER :=&input_out;
new   3:    v_out    NUMBER :=72;
```

pgrocer@cisweb:~

```
SQL> desc inout_table;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 COL_IN                                             NUMBER
 COL_OUT                                            NUMBER
 COL_INOUT                                          NUMBER

SQL> delete from inout_table;

2 rows deleted.

SQL> select * from inout_table;

no rows selected

SQL> edit calc_proc_calc

SQL> edit proc_calc

SQL> @ proc_calc

Procedure created.

SQL>
```

Title: Nov 14-10:03 AM (8 of 20)

File Edit View History Bookmarks Tools Help

Introduction to procedures

www.pgrocer.net/Cis50/procedures.html

```
COL_INOUT                                NUMBER
```

In the example below, there is a procedure and a calling PL/SQL block. The calling block takes in three numbers and passes then to the procedure. In the sample run, I inputed 15, 72 and 99. These three numbers were passed to the proc_calc procedure. The input number is passed on a read only basis so I did no processing with the p_in. The output number is passed on a write only basis and so the 72 is ignored and p_out is calculated within the program as 15 + 15. The input/output number comes in as 99 but within the program I add 15 + 15 + 15 so this is the number that appears on the table through the insert.

SQL CODE:

```
SQL> edit proc_calc
```

PL/SQL CODE:

```
CREATE OR REPLACE PROCEDURE proc_calc
   (p_in      IN NUMBER,
    p_out     OUT NUMBER,
    p_inout   IN OUT NUMBER) IS
BEGIN
    p_out := p_in + p_in;
    p_inout := p_in + p_in + p_in;
    INSERT INTO inout_table (col_in, col_out, col_inout)
      VALUES (p_in, p_out, p_inout);
END;
/
```

SQL CODE:

```
SQL> @ proc_calc

Procedure created.

SQL> edit calc_proc_calc
```

PL/SQL CODE:

```
DECLARE
   v_in    NUMBER :=&input_in;
   v_out   NUMBER :=&input_out;
   v_inout NUMBER :=&input_inout;
BEGIN
   proc_calc(v_in, v_out, v_inout);
END;
/
```

SQL CODE:

```
SQL> @ calc_proc_calc

Input truncated to 1 characters
Enter value for input_in: 15
old   2:    v_in    NUMBER :=&input_in;
new   2:    v_in    NUMBER :=15;
Enter value for input_out: 72
old   3:    v_out   NUMBER :=&input_out;
new   3:    v_out   NUMBER :=72;
```

pgrocer@cisweb:~

```
SQL> @ proc_calc

Procedure created.

SQL> @ calc_proc_calc
Enter value for input_in: 17
old   2:    v_in NUMBER :=&input_in;
new   2:    v_in NUMBER :=17;
Enter value for input_out: 25
old   3:    v_out NUMBER :=&input_out;
new   3:    v_out NUMBER :=25;
Enter value for input_inout: 53
old   4:    v_inout NUMBER :=&input_inout;
new   4:    v_inout NUMBER :=53;

PL/SQL procedure successfully completed.

SQL> select * from inout_table;

    COL_IN    COL_OUT  COL_INOUT
---------- ---------- ----------
        17                    53

SQL>
```

Type here to search

10:04 AM 11/14/2019

Title: Nov 14-10:04 AM (9 of 20)

File Edit View History Bookmarks Tools Help

Introduction to procedures

www.pgrocer.net/Cis50/procedures.html

```
33333 300 10-MAR-99       10
22222 100 14-MAR-99       10
12121 100 04-JUN-99       50
11111 200 12-JUN-99       35
23456 300 14-JUN-99       10
12121 300 10-JUN-99       75

9 rows selected.
```

Going a little further, we find that there are three modes for formal parameters: IN, OUT or IN OUT. IN sends the parameters from the calling program to the procedure. OUT sends the parameters from the procedure back to the calling program. IN OUT passes to and receives from the procedure. The default mode is IN. Since no mode was s pecified in the example above, the default was IN. To test this, I created the following table:

SQL CODE:

```
SQL> CREATE TABLE inout_table
  2  (col_in NUMBER, col_out NUMBER, col_inout NUMBER);

Table created.

SQL> DESC inout_table
 Name                            Null?    Type
 ------------------------------- -------- ----
 COL_IN                                   NUMBER
 COL_OUT                                  NUMBER
 COL_INOUT                                NUMBER
```

In the example below, there is a procedure and a calling PL/SQL block. The calling block takes in three numbers and passes then to the procedure. In the sample run, I inputed 15, 72 and 99. These three numbers were passed to the proc_calc procedure. The input number is passed on a read only basis so I did no processing with the p_in. The output number is passed on a write only basis and so the 72 is ignored and p_out is calculated within the program as 15 + 15. The input/output number comes in as 99 but within the program I add 15 + 15 + 15 so this is the number that appears on the table through the insert.

SQL CODE:

```
SQL> edit proc_calc
```

PL/SQL CODE:

```
CREATE OR REPLACE PROCEDURE proc_calc
   (p_in     IN NUMBER,
    p_out    OUT NUMBER,
    p_inout  IN OUT NUMBER) IS
BEGIN
    p_out := p_in + p_in;
    p_inout := p_in + p_in + p_in;
    INSERT INTO inout_table (col_in, col_out, col_inout)
      VALUES (p_in, p_out, p_inout);
END;
/
```

SQL CODE:

```
SQL> @ proc_calc

Procedure created.
```

Type here to search

10:08 AM 11/14/2019

```
Procedure created.

SQL> @ calc_proc_calc
Enter value for input_in: 17
old   2:    v_in NUMBER :=&input_in;
new   2:    v_in NUMBER :=17;
Enter value for input_out: 25
old   3:    v_out NUMBER :=&input_out;
new   3:    v_out NUMBER :=25;
Enter value for input_inout: 53
old   4:    v_inout NUMBER :=&input_inout;
new   4:    v_inout NUMBER :=53;

PL/SQL procedure successfully completed.

SQL> select * from inout_table;

    COL_IN    COL_OUT  COL_INOUT
---------- ---------- ----------
        17                    53
        17         34         51

SQL>
```

File Edit View History Bookmarks Tools Help

Introduction to procedures

www.pgrocer.net/Cis50/procedures.html

```
Enter value for input_inout: 99
old   4:     v_inout NUMBER :=&input_inout;
new   4:     v_inout NUMBER :=99;

PL/SQL procedure successfully completed.

SQL> SELECT * FROM inout_table;


   COL_IN   COL_OUT COL_INOUT
--------- --------- ---------
       15        30        45
```

I then altered the procedure (proc_calc) to add the input that was received into p_inout and the variable that was received in p_in and store the result in p_inout. I inputed 23, 33, 43 and the output acted as expected and added the 43 +23 to get 66.

PL/SQL CODE:

```
CREATE OR REPLACE PROCEDURE proc_calc
   (p_in      IN NUMBER,
    p_out     OUT NUMBER,
    p_inout   IN OUT NUMBER) IS
BEGIN
    p_out := p_in + p_in;
    p_inout := p_inout + p_in;
    INSERT INTO inout_table (col_in, col_out, col_inout)
      VALUES (p_in, p_out, p_inout);
END;
/
```

SQL CODE:

```
SQL> @ calc_proc_calc
Input truncated to 1 characters
Enter value for input_in: 23
old   2:     v_in     NUMBER :=&input_in;
new   2:     v_in     NUMBER :=23;
Enter value for input_out: 33
old   3:     v_out    NUMBER :=&input_out;
new   3:     v_out    NUMBER :=33;
Enter value for input_inout: 43
old   4:     v_inout NUMBER :=&input_inout;
new   4:     v_inout NUMBER :=43;

PL/SQL procedure successfully completed.

SQL> SELECT * FROM inout_table;

   COL_IN   COL_OUT COL_INOUT
--------- --------- ---------
       15        30        45
       23        46        66
```

I then went in and altered proc_calc to use p_out in the calculation of p_out as I had done with p_inout. This time when I did the @ proc_calc I got a warning telling me that the procedure was created with compilation errors. This was because I misused the field defined as OUT which is write only. When I tried to execute the calling procedure I received errors.

Type here to search

10:12 AM
11/14/2019

File Edit View History Bookmarks Tools Help

Introduction to procedures

www.pgrocer.net/Cis50/procedures.html

```
old   2:     v_in     NUMBER :=&input_in;
new   2:     v_in     NUMBER :=23;
Enter value for input_out: 33
old   3:     v_out    NUMBER :=&input_out;
new   3:     v_out    NUMBER :=33;
Enter value for input_inout: 43
old   4:     v_inout NUMBER :=&input_inout;
new   4:     v_inout NUMBER :=43;

PL/SQL procedure successfully completed.

SQL> SELECT * FROM inout_table;

   COL_IN   COL_OUT COL_INOUT
--------- --------- ---------
       15        30        45
       23        46        66
```

I then went in and altered proc_calc to use p_out in the calculation of p_out as I had done with p_inout. This time when I did the @ proc_calc I got a warning telling me that the procedure was created with compilation errors. This was because I misused the field defined as OUT which is write only. When I tried to execute the calling procedure I received errors.

SQL CODE:

```
SQL> edit proc_calc
```

PL/SQL CODE:

```
CREATE OR REPLACE PROCEDURE proc_calc
   (p_in      IN NUMBER,
    p_out     OUT NUMBER,
    p_inout   IN OUT NUMBER) IS
BEGIN
    p_out := p_out + p_in;
    p_inout := p_inout + p_in;
    INSERT INTO inout_table (col_in, col_out, col_inout)
      VALUES (p_in, p_out, p_inout);
END;
/
```

SQL CODE:

```
SQL> @ proc_calc
Input truncated to 1 characters

Warning: Procedure created with compilation errors.

SQL> @ calc_proc_calc
Input truncated to 1 characters
Enter value for input_in: 57
old   2:     v_in     NUMBER :=&input_in;
new   2:     v_in     NUMBER :=57;
Enter value for input_out: 45
old   3:     v_out    NUMBER :=&input_out;
new   3:     v_out    NUMBER :=45;
Enter value for input_inout: 89
old   4:     v_inout NUMBER :=&input_inout;
```

Type here to search

10:15 AM
11/14/2019

File Edit View History Bookmarks Tools Help

Introduction to procedures

www.pgrocer.net/Cis50/procedures.html

```
        23         46         66
```

I then went in and altered proc_calc to use p_out in the calculation of p_out as I had done with p_inout. This time when I did the @ proc_calc I got a warning telling me that the procedure was created with compilation errors. This was because I misused the field defined as OUT which is write only. When I tried to execute the calling procedure I received errors.

SQL CODE:

```
SQL> edit proc_calc
```

PL/SQL CODE:

```
CREATE OR REPLACE PROCEDURE proc_calc
   (p_in      IN NUMBER,
    p_out     OUT NUMBER,
    p_inout   IN OUT NUMBER) IS
BEGIN
    p_out := p_out + p_in;
    p_inout := p_inout + p_in;
    INSERT INTO inout_table (col_in, col_out, col_inout)
      VALUES (p_in, p_out, p_inout);
END;
/
```

SQL CODE:

```
SQL> @ proc_calc
Input truncated to 1 characters

Warning: Procedure created with compilation errors.

SQL> @ calc_proc_calc
Input truncated to 1 characters
Enter value for input_in: 57
old   2:     v_in     NUMBER :=&input_in;
new   2:     v_in     NUMBER :=57;
Enter value for input_out: 45
old   3:     v_out    NUMBER :=&input_out;
new   3:     v_out    NUMBER :=45;
Enter value for input_inout: 89
old   4:     v_inout NUMBER :=&input_inout;
new   4:     v_inout NUMBER :=89;
   proc_calc(v_in, v_out, v_inout);
   *
ERROR at line 6:
ORA-06550: line 6, column 4:
PLS-00905: object SCOTT.PROC_CALC is invalid
ORA-06550: line 6, column 4:
PL/SQL: Statement ignored
```

pgrocer@cisweb:~

```
Enter value for input_out: 12
old   3:    v_out NUMBER :=&input_out;
new   3:    v_out NUMBER :=12;
Enter value for input_inout: 12
old   4:    v_inout NUMBER :=&input_inout;
new   4:    v_inout NUMBER :=12;

PL/SQL procedure successfully completed.

SQL> select * from inout_table;

    COL_IN    COL_OUT  COL_INOUT
---------- ---------- ----------
        17                    53
        17         34         51
        12                    36

SQL> edit proc_calc

SQL> @ proc_calc

Procedure created.

SQL> @ calc_proc_calc
```

Next, I changed the proc_calc back and changed the calling program calc_proc_calc to include an INSERT to write a record to the table. This shows the information that comes back from the procedure to the calling program.

SQL CODE:

```
SQL> edit proc_calc
```

Type here to search

10:19 AM 11/14/2019

Title: Nov 14-10:19 AM (14 of 20)

File Edit View History Bookmarks Tools Help

CIS150/50 Course Page

www.pgrocer.net/PFGCIS150.html

Oracle and SQL
CIS150/50

About This Course:
- **Need help or support?**
- **Introduction to CIS150**
- **Syllabus**
- **Withdrawal Policy as of Fall 2017**

Site Resources:
- **Notes and Handouts**
- **Presentations**
- **SmartBoard Presentations**
- **Audio**

Weekly Schedule:
- **Weekly schedule - CIS150/50**

Using Oracle:
- **Installation and login**

Links:
- **Links & Tutorials**

Week #10 Week of November 4th	**Another set of cursor examples** **Covers: Another set of cursor exampes** **Separate speaker notes for Another set of cursor examples** Examples we are going over in class: **payprog.sql** **payprog1.sql** **payprog2.sql** **data.txt** Data layout for studentchk.sql **studentchk.sql** **createdonor.sql** Script to create donor **donorcursor1.sql** **donorcursor2.sql** **donorcursor3.sql** **decodecursor1.sql** The plan for Thursday was: In class we will start a cursor program. I want you to write a script that will handle the inven table and the department table. The script should accumulate the onhand for departments and write to a new table where you will store the dept, the department name and the total of the onhand. I would also like you to use the dbms lines to show the processing that happens. Everyone needs to turn in this script including the dbms output and a listing of what is on the new table. It is now an assignment we can work next Thursday. **Next assignment:** Use cursors to solve these problems. First Problem:Write a PL/SQL program to accomplish the following: Using the sample emp database, give everyone in dept 10 a 5% raise, everyone in dept 20 a 7.5% raise, everyone in department dept 30 a 10% raise. Do this using a simple IF...ELSE and a loop. Second Problem:Modify the problem in #1 to meet the following criteria: IF dept is 10 and the salary is greater than 2000 give a 6% raise otherwise give a 7% raise. IF dept is 20 and the salary is greater than 2500 give a 5% raise otherwise give a 5.5% raise. IF dept is 30 and the salary is greater than 1000 give a 7% raise otherwise give a 6.5% raise. Third Problem: Set up an inventory table and a transaction table that has sales, returns, and purchases (the transaction table should have a code with S for sales, R for returns and P for purchases). Create scripts to create and insert data into these tables. Fourth Problem: Using the tables you created in the fourth problem, process the transactions and determine the impact on inventory. Display information that gives the original inventory and the inventory after the sales, returns and purchases have been processed. You need to use cursors. Fourth Problem: I want you to redo this program so that the EXIT does not come in the middle of code but rather at the bottom. Do the second version of the program on this example sheet: **Example**
Week #9 Week of October 28th	We will look at the loops and data manipulation in PL/SQL that are listed under last week. Then we will continue on with PL/SQL. Returned to finish this handout: **Introduction to PL/SQL** Note the accompanying Powerpoint is listed on week #8. Then started cursors: **Introduction to Cursors** **Example 1 using cursors** **Covers: Cursors in PL/SQL, cursor example and continuation of first cursor example** **Separate speaker notes for Cursors in PL/SQL, cursor example and continuation of first cursor example** ***Assignments:*** **Oracle Loop Assignment** **Oracle PL/SQL Intro Quiz** - Start in class
	Thursday is an open lab day from 9:30 until 5:00 in K101. Come for as many hours as you can and work to get caught up with Joe or myself to answer questions. We will start PL/SQL this week. This week, I am going to look at The Introduction to PL/SQL (the beginning part), skip for a while the More examples of input/output procedures (we will go back to them) and work on Introduction to IF statements. The things below will probably be partially done and the rest we will look at next week. **Introduction to PL/SQL** **Introduction to IF statements** **More on PL/SQL IF statements** **Introduction to loops**

Type here to search

10:23 AM
11/14/2019

Title: Nov 14-10:23 AM (15 of 20)

File Edit View History Bookmarks Tools Help

Continuation of first cursor exampl × | CIS150/50 Course Page ×

www.pgrocer.net/PFGCIS150.html

Oracle and SQL
CIS150/50

About This Course:

Need help or support?
Introduction to CIS150
Syllabus
Withdrawal Policy as of Fall 2017

Site Resources:

Notes and Handouts
Presentations
SmartBoard Presentations
Audio

Weekly Schedule:

Weekly schedule - CIS150/50

Using Oracle:

Installation and login

Links:

Links & Tutorials

Week #10
Week of November 4th

Output
deptno
deptname
tot on hand
111
first
1060

Note: The last date for student generated withdrawal is Thursday, November 14, 2019. Remember I cannot withdraw you so if you do not have the grades needed to pass the course, I will have to give you an F.
Continuing with cursor discussion we started last week.
We will continue with cursors and then back up to see some the the things under Notes and handouts that we did not cover yet.
Continuing example 1 using cursors
Another set of cursor examples
Covers: Another set of cursor examples
Separate speaker notes for Another set of cursor examples
Examples we are going over in class:
payprog.sql
payprog1.sql
payprog2.sql
data.txt Data layout for studentchk.sql
studentchk.sql
createdonor.sql Script to create donor
donorcursor1.sql
donorcursor2.sql
donorcursor3.sql
decodecursor1.sql
The plan for Thursday was:
In class we will start a cursor program. I want you to write a script that will handle the inven table and the department table. The script should accumulate the onhand for departments and write to a new table where you will store the dept, the department name and the total of the onhand. I would also like you to use the dbms lines to show the processing that happens. Everyone needs to turn in this script including the dbms output and a listing of what is on the new table. It is now an assignment we can work next Thursday.
Next assignment: Use cursors to solve these problems.
First Problem:Write a PL/SQL program to accomplish the following: Using the sample emp database, give everyone in dept 10 a 5% raise, everyone in dept 20 a 7.5% raise, everyone in department dept 30 a 10% raise. Do this using a simple IF...ELSE and a loop.
Second Problem:Modify the problem in #1 to meet the following criteria: IF dept is 10 and the salary is greater than 2000 give a 6% raise otherwise give a 7% raise. IF dept is 20 and the salary is greater than 2500 give a 5% raise otherwise give a 5.5% raise. IF dept is 30 and the salary is greater than 1000 give a 7% raise otherwise give a 6.5% raise.
Third Problem: Set up an inventory table and a transaction table that has sales, returns, and purchases (the transaction table should have a code with S for sales, R for returns and P for purchases). Create scripts to create and insert data into these tables.
Fourth Problem: Using the tables you created in the fourth problem, process the transactions and determine the impact on inventory. Display information that gives the original inventory and the inventory after the sales, returns and purchases have been processed. You need to use cursors.
Fourth Problem: I want you to redo this program so that the EXIT does not come in the middle of code but rather at the bottom. Do the second version of the program on this example sheet: **Example**

dept
deptno
deptname
onhand

111
first
1000

inven
Item no
date
amt received
deptno

123 11/19 20 111
123 11/20 40 111

Week #9
Week of October 28th

We will look at the loops and data manipulation in PL/SQL that are listed under last week. Then we will continue on with PL/SQL.
Returned to finish this handout: **Introduction to PL/SQL** Note the accompanying Powerpoint is listed on week #8.
Then started cursors:
Introduction to Cursors
Example 1 using cursors
Covers: Cursors in PL/SQL, cursor example and continuation of first cursor example
Separate speaker notes for Cursors in PL/SQL, cursor example and continuation of first cursor example
Assignments:
Oracle Loop Assignment
Oracle PL/SQL Intro Quiz - Start in class

Thursday is an open lab day from 9:30 until 5:00 in K101. Come for as many hours as you can and work to get caught up with Joe or myself to answer questions.
We will start PL/SQL this week.
This week, I am going to look at The Introduction to PL/SQL (the beginning part), skip for a while the More examples of input/output procedures (we will go

Type here to search

10:31 AM
11/14/2019

Title: Nov 14-10:31 AM (16 of 20)

SQL CODE:

```
SET SERVEROUTPUT ON
DECLARE
  v_drive_no      drive.driveno%TYPE;
  v_drive_name    drive.drivename%TYPE;
  v_contamt       donation.contamt%TYPE;
  v_tot_contamt   cont_info.contamt%TYPE;
  CURSOR drive_cursor IS
     SELECT driveno, drivename FROM drive
     ORDER BY driveno;
  CURSOR donation_cursor IS
     SELECT contamt FROM donation
     WHERE v_drive_no = driveno
     ORDER BY driveno;
  BEGIN
     OPEN drive_cursor;
     FETCH drive_cursor INTO v_drive_no, v_drive_name;
     WHILE drive_cursor%FOUND LOOP
       IF donation_cursor%ISOPEN THEN
         CLOSE donation_cursor;
       END IF;
       OPEN donation_cursor;
       v_tot_contamt := 0;
       FETCH donation_cursor INTO v_contamt;
       WHILE donation_cursor%FOUND LOOP
         v_tot_contamt := v_tot_contamt + v_contamt;
         dbms_output.put_line('The current amount is: '||v_tot_contamt);
         FETCH donation_cursor INTO v_contamt;
       END LOOP;
       INSERT into cont_info
          VALUES(v_drive_no, v_drive_name, v_tot_contamt);
       CLOSE donation_cursor;
       FETCH drive_cursor INTO v_drive_no, v_drive_name;
     END LOOP;
     CLOSE drive_cursor;
END;
/
SET SERVEROUTPUT OFF
```

SQL CODE:

```
SQL> @ cursor7b
The current amount is: 25
The current amount is: 45
The current amount is: 55
The current amount is: 105
The current amount is: 40
The current amount is: 75
The current amount is: 10
The current amount is: 20

PL/SQL procedure successfully completed.

Input truncated to 20 characters
SQL> SELECT * FROM cont_info;

DRI DRIVENAME            CONTAMT
```

Output
deptno
deptname
tot on hand
111
first
1062
60
cont_info

dept
deptno
deptname
on hand
111
first
1000
optional
drive

123 11/19 20 111
123 11/20 40 111
Inven
Item no
date
amt received
deptno
donation

File Edit View History Bookmarks Tools Help

Continuation of first cursor exampl × | CIS150/50 Course Page × | CIS150/50 Course Page ×

www.pgrocer.net/PFGCIS150.html

Oracle and SQL
CIS150/50

About This Course:
- Need help or support?
- Introduction to CIS150
- Syllabus
- Withdrawal Policy as of Fall 2017

Site Resources:
- Notes and Handouts
- Presentations
- SmartBoard Presentations
- Audio

Weekly Schedule:
- Weekly schedule - CIS150/50

Using Oracle:
- Installation and login

Links:
- Links & Tutorials

Week #10
Week of November 4th

I will not be in on November 7th. Joe will be available starting at 12:30 to 5 for an open lab.
A group of alum are coming to my CIS120 classes on November 5th to talk about jobs and careers in IT. Please try to come. Note they will come to both the 11:00 class and the 12:30 class.
Note: The last date for student generated withdrawal is Thursday, November 14, 2019. Remember I cannot withdraw you so if you do not have the grades needed to pass the course, I will have to give you an F.
Continuing with cursor discussion we started last week.
We will continue with cursors and then back up to see some the the things under Notes and handouts that we did not cover yet.
Continuing example 1 using cursors
Another set of cursor examples
Covers: Another set of cursor exampes
Separate speaker notes for Another set of cursor examples
Examples we are going over in class:
payprog.sql
payprog1.sql
payprog2.sql
data.txt Data layout for studentchk.sql
studentchk.sql
createdonor.sql Script to create donor
donorcursor1.sql
donorcursor2.sql
donorcursor3.sql
decodecursor1.sql
The plan for Thursday was:
In class we will start a cursor program. I want you to write a script that will handle the inven table and the department table. The script should accumulate the onhand for departments and write to a new table where you will store the dept, the department name and the total of the onhand. I would also like you to use the dbms lines to show the processing that happens. Everyone needs to turn in this script including the dbms output and a listing of what is on the new table. It is now an assignment we can work next Thursday.
Next assignment: Use cursors to solve these problems.
First Problem:Write a PL/SQL program to accomplish the following: Using the sample emp database, give everyone in dept 10 a 5% raise, everyone in dept 20 a 7.5% raise, everyone in department dept 30 a 10% raise. Do this using a simple IF...ELSE and a loop.
Second Problem:Modify the problem in #1 to meet the following criteria: IF dept is 10 and the salary is greater than 2000 give a 6% raise otherwise give a 7% raise. IF dept is 20 and the salary is greater than 2500 give a 5% raise otherwise give a 5.5% raise. IF dept is 30 and the salary is greater than 1000 give a 7% raise otherwise give a 6.5% raise.
Third Problem: Set up an inventory table and a transaction table that has sales, returns, and purchases (the transaction table should have a code with S for sales, R for returns and P for purchases). Create scripts to create and insert data into these tables.
Fourth Problem: Using the tables you created in the fourth problem, process the transactions and determine the impact on inventory. Display information that gives the original inventory and the inventory after the sales, returns and purchases have been processed. You need to use cursors.
Fourth Problem: I want you to redo this program so that the EXIT does not come in the middle of code but rather at the bottom. Do the second version of the program on this example sheet: **Example**

Week #9
Week of October 28th

We will look at the loops and data manipulation in PL/SQL that are listed under last week. Then we will continue on with PL/SQL.
Returned to finish this handout: **Introduction to PL/SQL** Note the accompanying Powerpoint is listed on week #8.
Then started cursors:
Introduction to Cursors
Example 1 using cursors
Covers: Cursors in PL/SQL, cursor example and continuation of first cursor example
Separate speaker notes for Cursors in PL/SQL, cursor example and continuation of first cursor example
Assignments:
Oracle Loop Assignment
Oracle PL/SQL Intro Quiz - Start in class

Thursday is an open lab day from 9:30 until 5:00 in K101. Come for as many hours as you can and work to get caught up with Joe or myself to answer

123 11/19 20 111
123 11/20 40 111
Inven
Item no
date
amt received
deptno

Dept
deptno
deptname
onhand

111
first
1000

optional

Output
deptno
deptname
tot on hand
111
first
1060
60

continues

drive

donation

inven
dept
111 first
111 20
111 40

Type here to search

10:44 AM
11/14/2019

Title: Nov 14-10:44 AM (18 of 20)

File Edit View History Bookmarks Tools Help

Continuation of first cursor exampl | CIS150/50 Course Page | CIS150/50 Course Page

www.pgrocer.net/PFGCIS150.html

Oracle and SQL CIS150/50

About This Course:

- **Need help or support?**
- **Introduction to CIS150**
- **Syllabus**
- **Withdrawal Policy as of Fall 2017**

Site Resources:

- **Notes and Handouts**
- **Presentations**
- **SmartBoard Presentations**
- **Audio**

Weekly Schedule:

- **Weekly schedule - CIS150/50**

Using Oracle:

- **Installation and login**

Links:

- **Links & Tutorials**

Week #10 Week of November 4th Output deptno deptname tot on hand 111 first 1060 60 cont-inue	I will not be in on November 7th. Joe will be available starting at 12:30 to 5 for an open lab. A group of alum are coming to my CIS120 classes on November 5th to talk about jobs and careers in IT. Please try to come. Note they will come to both the 11:00 class and the 12:30 class. Note: The last date for student generated withdrawal is Thursday, November 14, 2019. Remember I cannot withdraw you so if you do not have the grades needed to pass the course, I will have to give you an F. Continuing with cursor discussion we started last week. We will continue with cursors and then back up to see some the the things under Notes and handouts that we did not cover yet. **Continuing example 1 using cursors** **Another set of cursor examples** **Covers: Another set of cursor exampes** **Separate speaker notes for Another set of cursor examples** Examples we are going over in class: **payprog.sql** **payprog1.sql** **payprog2.sql** **data.txt** Data layout for studentchk.sql **studentchk.sql** **createdonor.sql** Script to create donor **donorcursor1.sql** **donorcursor2.sql** **donorcursor3.sql** **decodecursor1.sql** The plan for Thursday was: In class we will start a cursor program. I want you to write a script that will handle the inven table and the department table. The script should accumulate the onhand for departments and write to a new table where you will store the dept, the department name and the total of the onhand. I would also like you to use the dbms lines to show the processing that happens. Everyone needs to turn in this script including the dbms output and a listing of what is on the new table. It is now an assignment we can work next Thursday. **Next assignment:** Use cursors to solve these problems. First Problem:Write a PL/SQL program to accomplish the following: Using the sample emp database, give everyone in dept 10 a 5% raise, everyone in dept 20 a 7.5% raise, everyone in department dept 30 a 10% raise. Do this using a simple IF...ELSE and a loop. Second Problem:Modify the problem in #1 to meet the following criteria: IF dept is 10 and the salary is greater than 2000 give a 6% raise otherwise give a 7% raise. IF dept is 20 and the salary is greater than 2500 give a 5% raise otherwise give a 5.5% raise. IF dept is 30 and the salary is greater than 1000 give a 7% raise otherwise give a 6.5% raise. Third Problem: Set up an inventory table and a transaction table that has sales, returns, and purchases (the transaction table should have a code with S for sales, R for returns and P for purchases). Create scripts to create and insert data into these tables. Fourth Problem: Using the tables you created in the fourth problem, process the transactions and determine the impact on inventory. Display information that gives the original inventory and the inventory after the sales, returns and purchases have been processed. You need to use cursors. Fourth Problem: I want you to redo this program so that the EXIT does not come in the middle of code but rather at the bottom. Do the second version of the program on this example sheet: **Example** 123 11/19 20 111 123 11/20 40 111 dept deptno deptname on hand 111 first 1000 inven Item no date amt received deptno optional drive donation inven dept 111 first 11 20 11 40
Week #9 Week of October 28th	We will look at the loops and data manipulation in PL/SQL that are listed under last week. Then we will continue on with PL/SQL. Returned to finish this handout: **Introduction to PL/SQL** Note the accompanying Powerpoint is listed on week #8. Then started cursors: **Introduction to Cursors** **Example 1 using cursors** **Covers: Cursors in PL/SQL, cursor example and continuation of first cursor example** **Separate speaker notes for Cursors in PL/SQL, cursor example and continuation of first cursor example** *Assignments:* **Oracle Loop Assignment** **Oracle PL/SQL Intro Quiz** - Start in class

Type here to search

10:45 AM 11/14/2019

Title: Nov 14-10:45 AM (19 of 20)

File Edit View History Bookmarks Tools Help

Continuation of first cursor exampl × | CIS150/50 Course Page × | CIS150/50 Course Page × | +

www.pgrocer.net/PFGCIS150.html

Oracle and SQL
CIS150/50

About This Course:

Need help or support?
Introduction to CIS150
Syllabus
Withdrawal Policy as of Fall 2017

Site Resources:

Notes and Handouts
Presentations
SmartBoard Presentations
Audio

Weekly Schedule:

Weekly schedule - CIS150/50

Using Oracle:

Installation and login

Links:

Links & Tutorials

Week #10 Week of November 4th	I will not be in on November 7th. Joe will be available starting at 12:30 to 5 for an open lab. A group of alum are coming to my CIS120 classes on November 5th to talk about jobs and careers in IT. Please try to come. Note they will come to both the 11:00 class and the 12:30 class. Note: The last date for student generated withdrawal is Thursday, November 14, 2019. Remember I cannot withdraw you so if you do not have the grades needed to pass the course, I will have to give you an F. Continuing with cursor discussion we started last week. We will continue with cursors and then back up to see some the the things under Notes and handouts that we did not cover yet. **Continuing example 1 using cursors** **Another set of cursor examples** **Covers: Another set of cursor exampes** **Separate speaker notes for Another set of cursor examples** Examples we are going over in class: **payprog.sql** **payprog1.sql** **payprog2.sql** **data.txt** Data layout for studentchk.sql **studentchk.sql** **createdonor.sql** Script to create donor **donorcursor1.sql** **donorcursor2.sql** **donorcursor3.sql** **decodecursor1.sql** The plan for Thursday was: In class we will start a cursor program. I want you to write a script that will handle the inven table and the department table. The script should accumulate the onhand for departments and write to a new table where you will store the dept, the department name and the total of the onhand. I would also like you to use the dbms lines to show the processing that happens. Everyone needs to turn in this script including the dbms output and a listing of what is on the new table. It is now an assignment we can work next Thursday. **Next assignment:** Use cursors to solve these problems. First Problem:Write a PL/SQL program to accomplish the following: Using the sample emp database, give everyone in dept 10 a 5% raise, everyone in dept 20 a 7.5% raise, everyone in department dept 30 a 10% raise. Do this using a simple IF...ELSE and a loop. Second Problem:Modify the problem in #1 to meet the following criteria: IF dept is 10 and the salary is greater than 2000 give a 6% raise otherwise give a 7% raise. IF dept is 20 and the salary is greater than 2500 give a 5% raise otherwise give a 5.5% raise. IF dept is 30 and the salary is greater than 1000 give a 7% raise otherwise give a 6.5% raise. Third Problem: Set up an inventory table and a transaction table that has sales, returns, and purchases (the transaction table should have a code with S for sales, R for returns and P for purchases). Create scripts to create and insert data into these tables. Fourth Problem: Using the tables you created in the fourth problem, process the transactions and determine the impact on inventory. Display information that gives the original inventory and the inventory after the sales, returns and purchases have been processed. You need to use cursors. Fourth Problem: I want you to redo this program so that the EXIT does not come in the middle of code but rather at the bottom. Do the second version of the program on this example sheet: **Example**
Week #9 Week of October 28th	We will look at the loops and data manipulation in PL/SQL that are listed under last week. Then we will continue on with PL/SQL. Returned to finish this handout: **Introduction to PL/SQL** Note the accompanying Powerpoint is listed on week #8. Then started cursors: **Introduction to Cursors** **Example 1 using cursors** **Covers: Cursors in PL/SQL, cursor example and continuation of first cursor example** **Separate speaker notes for Cursors in PL/SQL, cursor example and continuation of first cursor example** ***Assignments:*** **Oracle Loop Assignment** **Oracle PL/SQL Intro Quiz** - Start in class

Type here to search

10:45 AM
11/14/2019

Title: Nov 14-10:45 AM (20 of 20)