Midterm (CSC352, CSC452)

1:00 pm - 8:59 pm, February 7. 2022

Please submit your midterm exam on D2L as you used to submit your homework assignments. For this midterm, text format is preferred, and the formats of rtf, doc, docx are allowed.

CSC 352 and 452 students share the same questions to avoid confusing.

When grading, the instructor will be lenient to students in 352 as requested by university policy.

If you think there is any problem with a question, either send me email, or write down your comments on your exam. If that is my side problem, your score will not be affected.

| Part I. True or False (2 points \* 6 ) | Part II. Multiple Choices  (5 points \* 2) | Part III. Coding  ( 13 point \* 6 = 78 points) |
| --- | --- | --- |

Part I. Please choose True or False for following questions (2 points for each question)

1. In the numeric for loop, system takes care of the increasing of the dummy counter. In the cursor for loop, the system takes care of the “Fetch” command and advances the pointer from the current record to the next record. **TRUE**
2. A cursor may be defined with some parameters and these parameters can be IN or OUT mode. **TRUE**
3. The variables declared in the inner block can be referenced in the outer block. **FALSE**
4. The cursor attribute cursor\_name%rowcount returns the number (how many) of records already processed up to this moment. **TRUE**
5. Table\_name%ROWTYPE is a composite data type. It is useful to store the data retrieved from database table. **TRUE**
6. Programmer can use cursor\_variable for loop same as cursor for loop. **FALSE**

Part II. Multiple choices. (5 points for each question). Please choose one and only one which is the wrong statement, or the least correct.

1. Please choose one and only one which is the wrong statement, or the least correct. **D**
2. Generally, nested table and variable size array require calling the constructor function to initialize its variable.
3. A variable of Associative array type does exist when defined, that is different from the variable of nested table and variable size array.
4. In defining Associative array types, the syntax as “... index by varchar2 (10) ” is allowed.
5. **Nested tables are always dense, no deletion of internal elements is allowed.** FALSE
6. The method of LIMIT will return the maximal size for a variable size array.
7. Please choose one and only one which is the wrong statement, or the least correct. **E**
8. All system (internally) defined errors have its own error code (number), and an error message.
9. Once an Oracle system (internally) defined error occurs in a program, no matter that error has predefined name or not, system engine will raise the exception.
10. A user defined exceptions for business reason will not be automatically raised by Oracle System.
11. Among thousands system internally defined exceptions, there are around 23 are given names by the system. For these 23, the programmer may write correspondent handler in the exception section.
12. **Among thousands system internally defined exceptions, there are around 23 are given names by the system. For these 23 exceptions with name, the system will not raise the exception automatically once these errors occur in a program.** FALSE

Part III. (13 points for each question in this part, total 78 points)

Write short programs, you need to submit your codes and the OUTPUT of your program.

1. Following code uses numeric for loop to print out a matrix.

Please change the codes to use while loop (to print out the same result).

DECLARE

str varchar2 (80);

BEGIN

FOR i IN 1 .. 9 LOOP -- outer loop

FOR j IN 1 .. 9 LOOP -- inner loop

str := str || ' ' || i || j ;

END LOOP; -- end of inner loop

DBMS\_OUTPUT.PUT\_LINE ( '|' || str ||' |' );

-- print out one line

str := null; -- re-set the line to null for next line/next i

END loop;

END;

/

**SET SERVEROUTPUT ON;**

**DECLARE**

**str varchar2 (80);**

**i NUMBER :=1;**

**j NUMBER :=1;**

**BEGIN**

**WHILE i < 9 LOOP -- outer loop**

**WHILE j < 9 LOOP -- inner loop**

**str := str || ' ' || i || j ;**

**j:= j + 1;**

**END LOOP; -- end of inner loop**

**DBMS\_OUTPUT.PUT\_LINE ( '|' || str ||' |' );**

**-- print out one line**

**str := null; -- re-set the line to null for next line/next i**

**i := i + 1;**

**j := 1;**

**END loop;**

**END;**

**OUTPUT:**

**| 11 12 13 14 15 16 17 18 |**

**| 21 22 23 24 25 26 27 28 |**

**| 31 32 33 34 35 36 37 38 |**

**| 41 42 43 44 45 46 47 48 |**

**| 51 52 53 54 55 56 57 58 |**

**| 61 62 63 64 65 66 67 68 |**

**| 71 72 73 74 75 76 77 78 |**

**| 81 82 83 84 85 86 87 88 |**

**PL/SQL procedure successfully completed.**

1. Based on table employees.

In this question, you will write a PL/SQL block. The program will define a cursor with one parameters called deptno which has data type of number. The select statement will retrieve the Employee\_id, First\_name, Last\_name for those that their department\_id belongs to that deptno.

In the executable section, you will open that cursor two times, first time with the parameter of value as 20, please use basic LOOP.

The second time the parameter value will be 60, it requires to use cursor for loop.

Your program will print out the information you have retrieved.

SET SERVEROUTPUT ON;

DECLARE

CURSOR c\_emp ( p\_deptno number) IS

SELECT employee\_id, first\_name, last\_name

FROM employees

WHERE department\_id = p\_deptno;

fname employees.first\_name%TYPE;

lname employees.last\_name%TYPE;

eid employees.employee\_id%TYPE;

BEGIN

OPEN c\_emp(20); -- pass value to parameter

DBMS\_OUTPUT.PUT\_LINE('OUTPUT for deptno as 20: --- ');

LOOP

FETCH c\_emp INTO eid, fname, lname;

EXIT WHEN c\_emp%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('employee id ' || eid ||', name is '

|| RPAD(fname, 10)|| ' ' || RPAD(lname, 10)|| ';');

END LOOP;

CLOSE c\_emp;

OPEN c\_emp(60); -- pass value to parameter

DBMS\_OUTPUT.PUT\_LINE('OUTPUT for deptno as 60: --- ');

LOOP

FETCH c\_emp INTO eid, fname, lname;

EXIT WHEN c\_emp%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('employee id ' || eid ||', name is '

|| RPAD(fname, 10)|| ' ' || RPAD(lname, 10)|| ';');

END LOOP;

CLOSE c\_emp;

END;

**OUTPUT:**

OUTPUT for deptno as 20: ---

employee id 201, name is Michael Hartstein ;

employee id 202, name is Pat Fay ;

OUTPUT for deptno as 60: ---

employee id 103, name is Alexander Hunold ;

employee id 104, name is Bruce Ernst ;

employee id 105, name is David Austin ;

employee id 106, name is Valli Pataballa ;

employee id 107, name is Diana Lorentz ;

PL/SQL procedure successfully completed.

1. Based on table employees.

Assume that the company has decided a one-time bonus for the employees in department 100, the policy is as below.

For employees that have worked 26 years or more:

if salary > = 10000 then bonus := 1000 ,

if salary >= 7000 and salary < 10000 then bonus := 800 ,

if salary < 7000 then bonus := 600 .

For employees that have worked 24 years or more but less than 26 :

if salary > = 8000 then bonus := 700,

if salary >= 7000 and salary < 8000 then bonus := 600,

if salary < 7000 then bonus := 500.

For employees that have worked 23 years or more but less than 24 :

if salary > = 7000 then bonus := 700,

if salary >= 5000 and salary < 7000 then bonus := 600,

if salary < 5000 then bonus := 500.

All other employees in department 100 will get $400.

SET SERVEROUTPUT ON;

DECLARE

CURSOR c IS

SELECT employee\_id, salary, hire\_date

FROM employees

WHERE department\_id = 100

FOR UPDATE;

v c%ROWTYPE;

new\_sal employees.salary%TYPE;

date employees.hire\_date%type := sysdate;

BEGIN

OPEN c;

LOOP

FETCH c INTO v;

EXIT WHEN c%NOTFOUND;

IF (v.hire\_date - date) >= 26 THEN

IF v.salary >= 10000 THEN

new\_sal := v.salary + 1000 ;

ELSIF v.salary >= 7000 THEN

new\_sal := v.salary + 800 ;

ELSIF v.salary < 7000 THEN

new\_sal := v.salary + 600 ;

END IF;

ELSIF (v.hire\_date - date) >= 24 THEN

IF v.salary >= 8000 THEN

new\_sal := v.salary + 700 ;

ELSIF v.salary >= 7000 THEN

new\_sal := v.salary + 600 ;

ELSIF v.salary < 7000 THEN

new\_sal := v.salary + 500 ;

END IF;

ELSIF (v.hire\_date - date) >= 23 THEN

IF v.salary >= 7000 THEN

new\_sal := v.salary + 700 ;

ELSIF v.salary >= 5000 THEN

new\_sal := v.salary + 600 ;

ELSIF v.salary < 5000 THEN

new\_sal := v.salary + 500 ;

END IF;

ELSE new\_sal := v.salary + 400;

UPDATE employees

SET Salary = new\_sal

WHERE current of c;

DBMS\_OUTPUT.put\_line ( v.employee\_id ||' ' ||TO\_CHAR (v.salary, '$9,999') || ' '||

TO\_CHAR (new\_sal, '$9,999'));

END IF;

END LOOP;

CLOSE c;

rollback;

END;

**OUTPUT:**

108 ####### #######

109 $9,000 $9,400

110 $8,200 $8,600

111 $7,700 $8,100

112 $7,800 $8,200

113 $6,900 $7,300

PL/SQL procedure successfully completed.

Write an anonymous PL/SQL program, retrieve the info needed for each employee in department 100, calculate the amount of bonus each employee should get according the above policy. Your program will display each employee’s ID, the years worked, current salary and the bonus. Also accumulate the bonus from each employee, print out the total amount of the bonuses for department (ID) 100.

The expression

floor ( months\_between (sysdate, hire\_date) /12 )

may help you to calculate the work years for each employee.

SET SERVEROUTPUT ON;

DECLARE

CURSOR c IS

SELECT employee\_id, salary, hire\_date

FROM employees

WHERE department\_id = 100

FOR UPDATE;

v c%ROWTYPE;

new\_sal employees.salary%TYPE;

total\_bonus employees.salary%TYPE := 0;

BEGIN

OPEN c;

LOOP

FETCH c INTO v;

EXIT WHEN c%NOTFOUND;

IF (floor ( months\_between (sysdate, v.hire\_date) /12 )) >= 26 THEN

IF v.salary >= 10000 THEN

new\_sal := v.salary + 1000 ;

total\_bonus := total\_bonus + 1000;

ELSIF v.salary >= 7000 THEN

new\_sal := v.salary + 800 ;

total\_bonus := total\_bonus + 800;

ELSIF v.salary < 7000 THEN

new\_sal := v.salary + 600 ;

total\_bonus := total\_bonus + 600;

END IF;

ELSIF (floor ( months\_between (sysdate, v.hire\_date) /12 )) >= 24 THEN

IF v.salary >= 8000 THEN

new\_sal := v.salary + 700 ;

total\_bonus := total\_bonus + 700;

ELSIF v.salary >= 7000 THEN

new\_sal := v.salary + 600 ;

total\_bonus := total\_bonus + 600;

ELSIF v.salary < 7000 THEN

new\_sal := v.salary + 500 ;

total\_bonus := total\_bonus + 500;

END IF;

ELSIF (floor ( months\_between (sysdate, v.hire\_date) /12 )) >= 23 THEN

IF v.salary >= 7000 THEN

new\_sal := v.salary + 700 ;

total\_bonus := total\_bonus + 700;

ELSIF v.salary >= 5000 THEN

new\_sal := v.salary + 600 ;

total\_bonus := total\_bonus + 600;

ELSIF v.salary < 5000 THEN

new\_sal := v.salary + 500 ;

total\_bonus := total\_bonus + 500;

END IF;

ELSE new\_sal := v.salary + 400;

total\_bonus := total\_bonus + 400;

UPDATE employees

SET Salary = new\_sal

WHERE current of c;

DBMS\_OUTPUT.put\_line ( v.employee\_id ||' ' ||TO\_CHAR (v.salary, '$9,999') || ' '|| v.hire\_date || ' '|| TO\_CHAR (v.salary, '$9,999') || ' '||

TO\_CHAR (new\_sal, '$9,999')|| ' '|| total\_bonus);

END IF;

END LOOP;

CLOSE c;

rollback;

END;

**OUTPUT:**

113 $6,900 07-DEC-99 $6,900 $7,300 4200

PL/SQL procedure successfully completed.

1. One of the popular ways in exception handling is to keep a record of the error into a table for future review when needed. The code below will create a table to store the error information,

DROP TABLE log\_error; -- in case you have that table created, otherwise ignore it

CREATE TABLE log\_error (

Error\_date DATE DEFAULT SYSDATE,

Username VARCHAR2 (15) DEFAULT USER,

Err\_code NUMBER,

Err\_msg VARCHAR2 (255));

If your tables are created by running the script provided on D2L, then the department\_id in the Departments table is defined as unique.

First, please copy and run the above creating table code. Then, write a PL/SQL block that will run one SQL insert command, that will insert one new record into Departments table, with department\_id as 270, department\_name as 'Fortest', no manager\_id (or as null), location as 1700.

Your PL/SQL block should have an exception section, in which you will include the WHEN OTHERS handler, that will catch up unexpected errors. In that OTHERS handler, use SQLCODE and SQLERRM to catch the right error number and error message, inserting a record into the log\_error table when error occurs.

After you run the PL/SQL block, you need to run a SQL statement separately to display the contents of the log\_error table, attach the result as part of your output for this question.

DECLARE

Err\_code NUMBER;

Err\_msg VARCHAR2 (255);

sql\_stmt VARCHAR2(200);

BEGIN

sql\_stmt := 'INSERT INTO Departments(department\_id, department\_name, manager\_id, location\_id) VALUES (270, "Fortest", null, 1700 ) ' ;

EXECUTE IMMEDIATE sql\_stmt;

DBMS\_OUTPUT.PUT\_LINE('insertion is done, end of the program.');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE ('From OTHERS Exception, Error Code = ' ||

TO\_CHAR(SQLCODE)||'; Error Message: '|| SQLERRM);

sql\_stmt := 'INSERT INTO log\_error(error\_date, username, err\_code,

err\_msg) VALUES (sysdate, "mlakomy", SQLCODE, SQLERRM) ' ;

EXECUTE IMMEDIATE sql\_stmt;

END;

**output:**

From OTHERS Exception, Error Code = -984; Error Message: ORA-00984: column not allowed here

PL/SQL procedure successfully completed.

1. Define a strong cursor variable type, name it as DeptTyp. Its return type is departments%ROWTYPE, based on table departments. Then declare a cursor variable of this type.

In your program, first you will open the cursor variable for a select statement that will return all records that has no manager assigned to it yet, such as

SELECT \* FROM departments

WHERE manager\_id is null ;

Your program will print out the department ID, department names and the location\_id for these departments that have no manager assigned yet.

Second time, open that cursor variable again, this time for a select statement that will return those departments that their location ID is not 1700. (WHERE location\_id != 1700 ).

For this second time, your program will print out the department ID, department names and the manager ID for these departments.

DECLARE

TYPE deptTyp IS REF CURSOR RETURN departments%ROWTYPE;

cv deptTyp;

v departments%rowtype;

BEGIN

OPEN cv for

SELECT \* FROM departments WHERE manager\_id is null ;

LOOP

FETCH cv INTO v;

EXIT WHEN cv%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(v.department\_id || ': ' || v.department\_name || ' ' || v.location\_id );

END LOOP;

CLOSE cv;

DBMS\_OUTPUT.PUT\_LINE('second part:');

OPEN cv for

SELECT \* FROM departments WHERE location\_id != 1700 ;

LOOP

FETCH cv INTO v;

EXIT WHEN cv%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(v.department\_id || ': ' || v.department\_name || ' ' || v.manager\_id );

END LOOP;

CLOSE cv;

END;

**OUTPUT:**

120: Treasury 1700

130: Corporate Tax 1700

140: Control And Credit 1700

150: Shareholder Services 1700

160: Benefits 1700

170: Manufacturing 1700

180: Construction 1700

190: Contracting 1700

200: Operations 1700

210: IT Support 1700

220: NOC 1700

230: IT Helpdesk 1700

240: Government Sales 1700

250: Retail Sales 1700

260: Recruiting 1700

270: Payroll 1700

second part:

20: Marketing 201

40: Human Resources 203

50: Shipping 121

60: IT 103

70: Public Relations 204

80: Sales 145

PL/SQL procedure successfully completed.

1. Use table of Employees for this question. Write an anonymous PL/SQL block. Define a nested table type called Lastname\_list. That nested table will use employees’ last\_name as its elements, thus the data type of the elements will be varchar2 (25). Define a variable for this type. Your program will do,
2. Populate this nested table with the last\_name of the employees in the department (ID) 60.
3. Print out the contents of the nested table after having finished the population.
4. Display the value of limit of nested table.
5. Display the value of the last index.
6. Add one elements to the end this nested table, assign the value as ‘King’,
7. Delete the elements 2 and 3,
8. Display the current size (count) of this variable (after step 6),
9. Display the value of the last index,
10. Print out the existing elements of the nested table.

\*\*\* END of the file. \*\*

DECLARE

TYPE Lastname\_list IS Table OF employees.LAST\_NAME%TYPE ;

v Lastname\_list := Lastname\_list ();

counter integer := 1;

CURSOR c IS

SELECT last\_name

FROM employees

Where department\_id in (60) ;

BEGIN

FOR indx IN c LOOP

v.extend;

v (counter) := indx.last\_name;

counter := counter + 1 ;

END LOOP;

Counter := v.FIRST;

WHILE counter IS NOT NULL LOOP

DBMS\_OUTPUT.PUT\_LINE (RPAD ( counter, 5) || v (counter) );

counter := v.next (counter);

END LOOP;

IF v.limit is null THEN

DBMS\_OUTPUT.PUT\_LINE( CHR(10)|| 'Q3. For type NT, there is no Limit, '

|| 'the output of v.limit is Always null;');

ELSE

DBMS\_OUTPUT.PUT\_LINE( CHR(10)|| 'Q3. NT max size: '|| v.limit);

END IF;

DBMS\_OUTPUT.PUT\_LINE( CHR(10)|| 'Q4. The current last index value is: '|| v.last);

v.extend;

v(v.last) := 'King' ;

v.delete (2, 3) ;

DBMS\_OUTPUT.PUT\_LINE

(CHR (10) ||'Q7. After adding and deleting elements 2 and 3, the current size is: '||

v.count );

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Q8. The last index value is: '|| v.last );

DBMS\_OUTPUT.PUT\_LINE ( CHR(10)||'Q9. After add and deletion, the NT list as below:' );

Counter := v.FIRST;

DBMS\_OUTPUT.PUT\_LINE ('No Email ');

DBMS\_OUTPUT.PUT\_LINE ('---- -------------------');

WHILE counter IS NOT NULL LOOP

DBMS\_OUTPUT.PUT\_LINE (RPAD ( counter, 5) || v (counter) );

counter := v.next (counter);

END LOOP;

END;

**OUTPUT:**

1 Hunold

2 Ernst

3 Austin

4 Pataballa

5 Lorentz

Q3. For type NT, there is no Limit, the output of v.limit is Always null;

Q4. The current last index value is: 5

Q7. After adding and deleting elements 2 and 3, the current size is: 4

Q8. The last index value is: 6

Q9. After add and deletion, the NT list as below:

No Email

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

1 Hunold

4 Pataballa

5 Lorentz

6 King

PL/SQL procedure successfully completed.