Exercise 1. Write a nested cursor in which the parent cursor gathers information about each section of a

course. The child cursor counts the enrollment. The only output is one line for each course, with

the course name, section number, and total enrollment.

|  |  |
| --- | --- |
| SOLUTION: | OUTPUT: |
| SET SERVEROUTPUT ON  DECLARE  CURSOR parent\_cursor IS  SELECT course\_no, description  FROM course  WHERE course\_no != 0;  CURSOR child\_cursor(param\_course\_no IN course.course\_no%type) IS  SELECT st.section\_no, COUNT(\*) total\_per\_course  FROM section st, enrollment et  WHERE st.course\_no = param\_course\_no  AND st.section\_id = et.section\_id  GROUP BY st.section\_no;  BEGIN  FOR i IN parent\_cursor LOOP  FOR j IN child\_cursor(i.course\_no) LOOP  DBMS\_OUTPUT.PUT\_LINE(i.description||' ~~Section number '||j.section\_no||' has '||j.total\_per\_course||' students!~~');  END LOOP;  END LOOP;  END; | Data Bases 2 ~~Section number 10 has 2 students!~~  Statistics ~~Section number 30 has 1 students!~~  Computer Networks ~~Section number 80 has 1 students!~~  Software Engineering ~~Section number 90 has 1 students!~~ |

Exercise 2. Write an anonymous PL/SQL block that finds all the courses that have at least one section that is at its maximum enrollment. If no courses meet that criterion, pick two courses and create that

situation for each.

**a )** For each of those courses, add another section. The instructor for the new section should be

taken from the existing records in the instructor table. Use the instructor who is signed up to teach

the fewest courses. Handle the fact that, during the execution of your program, the instructor

teaching the most courses may change.

**b)** Use any exception-handling techniques you think are useful to capture error conditions.

|  |  |
| --- | --- |
| SOLUTION: | OUTPUT: |
| SET SERVEROUTPUT ON  DECLARE  intructor\_min\_courses instructor.instructor\_id%TYPE;  new\_section\_ID section.section\_id%TYPE;  new\_number\_section section.section\_no%TYPE := 0;  --create a cursor to point to the courses with at least one section at its capacity limit  --we need to use the enrollment and section tables which have in common "section\_id" column  CURSOR courses\_maxed IS  SELECT DISTINCT st.course\_no  FROM section st  WHERE st.capacity =(SELECT COUNT(section\_id)  FROM enrollment et  WHERE et.section\_id = st.section\_id);  BEGIN  FOR course\_maxed IN courses\_maxed LOOP  --we have a list with courses that have at least 1 full section  --for each we will add one new section who will be taught by the instructor with less courses  SELECT instructor\_id  INTO intructor\_min\_courses  FROM instructor  WHERE EXISTS( SELECT NULL  FROM section  WHERE section.instructor\_id = instructor.instructor\_id  GROUP BY instructor\_id  HAVING COUNT(\*) = (SELECT MIN(COUNT(\*))  FROM section  WHERE instructor\_id IS NOT NULL  GROUP BY instructor\_id)  )  AND ROWNUM = 1;    --we need to assign an ID to the new section that will be created  --we can take the maximum existing ID and maximum existing number and add some numeric integer (like 10)  SELECT MAX(section\_id)+10, MAX(section\_no)+10  INTO new\_section\_ID, new\_number\_section  FROM section;    INSERT INTO section(section\_id, course\_no, section\_no, instructor\_id, created\_by, created\_date, modified\_by, modified\_date)  VALUES (new\_section\_ID, course\_maxed.course\_no, new\_number\_section, intructor\_min\_courses, 'Kovaci', '07-April-2020', 'Kovaci', '07-April-2020');  COMMIT;  END LOOP;    EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('No data!');    WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Some errors!');  END; | Either the message from the exception statement or rows are inserted in the table |

Exercise 3. Construct 3 cursors. The first one, cursor c\_student takes no parameters and is a collection of students with a last name beginning with J. The second one c\_course takes in the parameter of

student\_ID to generate a list of courses that student is taking. The third one, c\_grade takes in two

parameters, section\_id and student\_id. In this way it can generate an average of the different grade

types (quizzes, homework, final, etc.) for that student for that course. Display the student name

for the first coursor. The second cursor takes the parameter of student\_id from the first cursor.

Only the description of the course is displayed. The third cursor takes in the parameter of

section\_id from the second cursor and student\_id from the first cursor. The grades are then

displayed.

|  |  |
| --- | --- |
| SOLUTION: | OUTPUT: |
| SET SERVEROUTPUT ON  DECLARE  CURSOR c\_student IS  SELECT \*  FROM student  WHERE last\_name LIKE 'P%';    CURSOR c\_course(param\_student\_ID IN student.student\_id%type) IS  SELECT ct.description, st.section\_id  FROM course ct, section st, enrollment et  WHERE et.student\_id = param\_student\_ID  AND ct.course\_no = st.course\_no  AND st.section\_id = et.section\_id;    CURSOR c\_grade(param\_section\_ID IN section.section\_id%type, param\_student\_ID IN student.student\_id%type) IS  SELECT gtype.description grd\_desc, TO\_CHAR (AVG(gt.numeric\_grade), '999.99') final\_grade  FROM enrollment et, grade gt, grade\_type gtype  WHERE et.section\_id = param\_section\_ID  AND et.student\_id = gt.student\_id  AND et.student\_id = param\_student\_ID  AND et.section\_id = gt.section\_id  AND gt.grade\_type\_code = gtype.grade\_type\_code  GROUP BY gtype.description;  BEGIN  FOR i IN c\_student LOOP  DBMS\_OUTPUT.PUT\_LINE(i.first\_name||' '||i.last\_name);  FOR j in c\_course(i.student\_id) LOOP  DBMS\_OUTPUT.PUT\_LINE(' ~~ '||j.description);  FOR k in c\_grade(j.section\_id, i.student\_id) LOOP  DBMS\_OUTPUT.PUT\_LINE(' ~~ '||k.final\_grade);  END LOOP;  END LOOP;  END LOOP;  END; | Ana Pup  ~~ Data Bases 2  ~~ 30.00  ~~ 30.00  ~~ 40.00  Emanuel Pop  ~~ Statistics  ~~ 30.00  ~~ 10.00  ~~ 60.00  Anita Pupu  ~~ Data Bases 2  Emanuel Popa  ~~ Statistics |