Name: Syed Noor Razi Ali
StudentID: 2070326

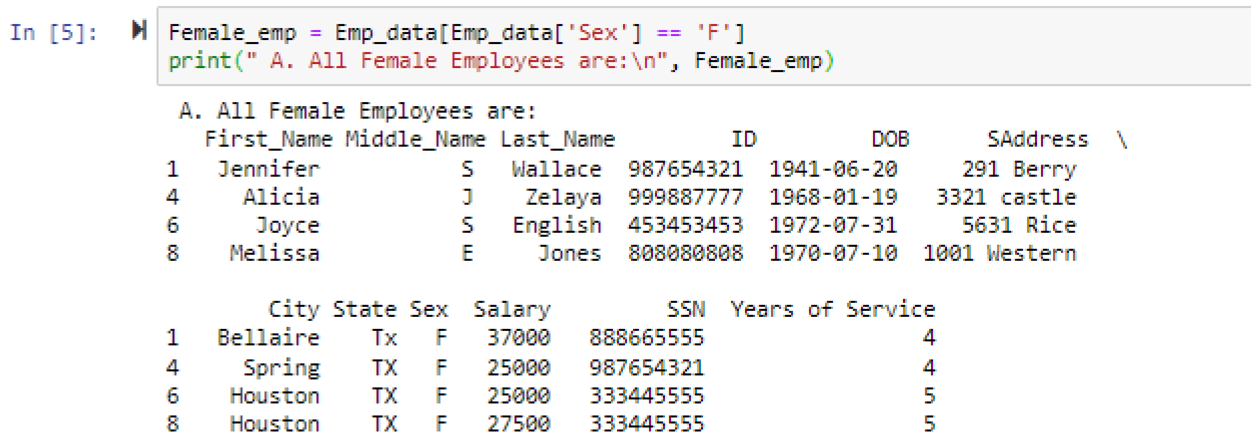
DSC 450: Database Processing for Large-Scale Analytics

Assignment Module 8

Part 1

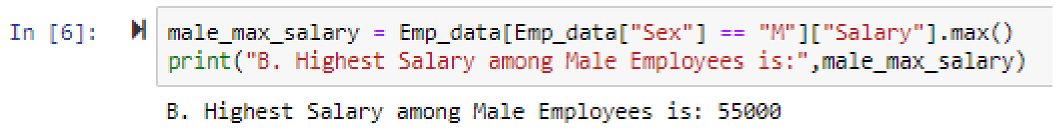
Use a DataFrame in python to define the following queries using the Employee data (employee.csv is attached). You can read it using pandas.read_csv('Employee.txt'). Adding optional parameter names=[] will allow you to rename the columns.

a. Find all female employees



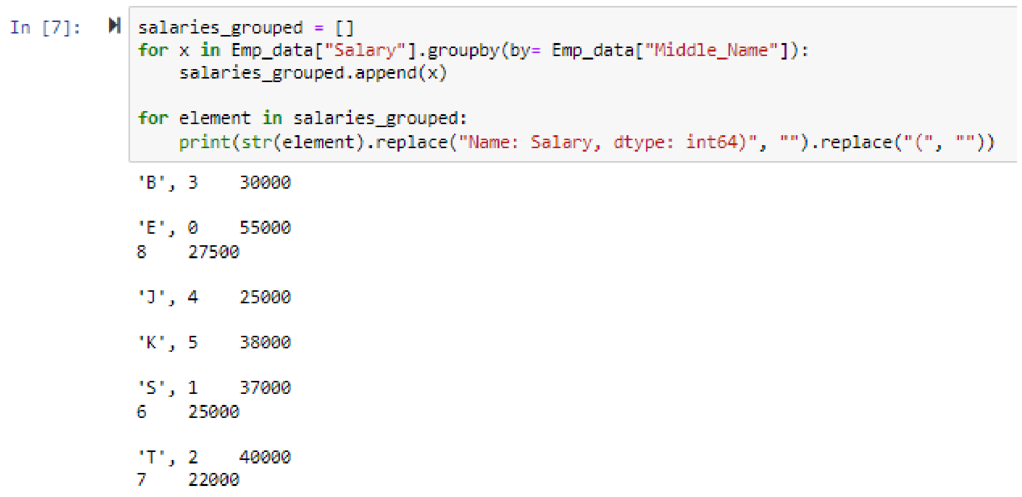
```
In [5]:  Female_emp = Emp_data[Emp_data['Sex'] == 'F']
         print(" A. All Female Employees are:\n", Female_emp)

 A. All Female Employees are:
   First_Name Middle_Name Last_Name         ID         DOB      SAddress  \
1   Jennifer           S   Wallace  987654321  1941-06-20     291 Berry
4     Alicia           J    Zelaya  999887777  1968-01-19   3321 castle
6      Joyce           S   English  453453453  1972-07-31     5631 Rice
8    Melissa           E     Jones  808080808  1970-07-10  1001 Western

       City State Sex  Salary        SSN  Years of Service
1  Bellaire    Tx   F   37000  888665555                 4
4    Spring    TX   F   25000  987654321                 4
6   Houston    TX   F   25000  333445555                 5
8   Houston    TX   F   27500  333445555                 5
```

b. Find the highest salary for male employees



```
In [6]:  male_max_salary = Emp_data[Emp_data["Sex"] == "M"]["Salary"].max()
         print("B. Highest Salary among Male Employees is:",male_max_salary)

         B. Highest Salary among Male Employees is: 55000
```

c. Print out salary groups (individual list of values without applying final aggregation) grouped by middle initial. That is, for each middle initial value, print all of the salaries in that group.



```
In [7]:  salaries_grouped = []
         for x in Emp_data["Salary"].groupby(by= Emp_data["Middle_Name"]):
             salaries_grouped.append(x)

         for element in salaries_grouped:
             print(str(element).replace("Name: Salary, dtype: int64)", "").replace("(", ""))
```

```
'B', 3    30000

'E', 0    55000
8    27500

'J', 4    25000

'K', 5    38000

'S', 1    37000
6    25000

'T', 2    40000
7    22000
```

Part 2

Consider the table STUDENT with attributes ID, Name, Midterm, Final, and Homework, and the table WEIGHTS with attributes MidPct, FinPct, and HWPct defined and populated by the following script:

```
DROP TABLE STUDENT CASCADE CONSTRAINTS;
CREATE TABLE STUDENT(
     ID         CHAR(3),
     Name       VARCHAR2(20),
     Midterm    NUMBER(3,0)    CHECK (Midterm>=0 AND Midterm<=100),
     Final           NUMBER(3,0)    CHECK (Final>=0 AND Final<=100),
     Homework   NUMBER(3,0)    CHECK (Homework>=0 AND Homework<=100),
     PRIMARY KEY (ID)
);
INSERT INTO STUDENT VALUES ( '445', 'Seinfeld', 86, 90, 99 );
INSERT INTO STUDENT VALUES ( '909', 'Costanza', 74, 72, 86 );
INSERT INTO STUDENT VALUES ( '123', 'Benes', 93, 89, 91 );
INSERT INTO STUDENT VALUES ( '111', 'Kramer', 99, 91, 93 );
INSERT INTO STUDENT VALUES ( '667', 'Newman', 78, 82, 84 );
INSERT INTO STUDENT VALUES ( '889', 'Banya', 50, 65, 50 );
SELECT * FROM STUDENT;

DROP TABLE WEIGHTS CASCADE CONSTRAINTS;
CREATE TABLE WEIGHTS(
     MidPct     NUMBER(2,0) CHECK (MidPct>=0 AND MidPct<=100),
     FinPct     NUMBER(2,0) CHECK (FinPct>=0 AND FinPct<=100),
     HWPct      NUMBER(2,0) CHECK (HWPct>=0 AND HWPct<=100)
);
INSERT INTO WEIGHTS VALUES ( 30, 30, 40 );
SELECT * FROM WEIGHTS;
```

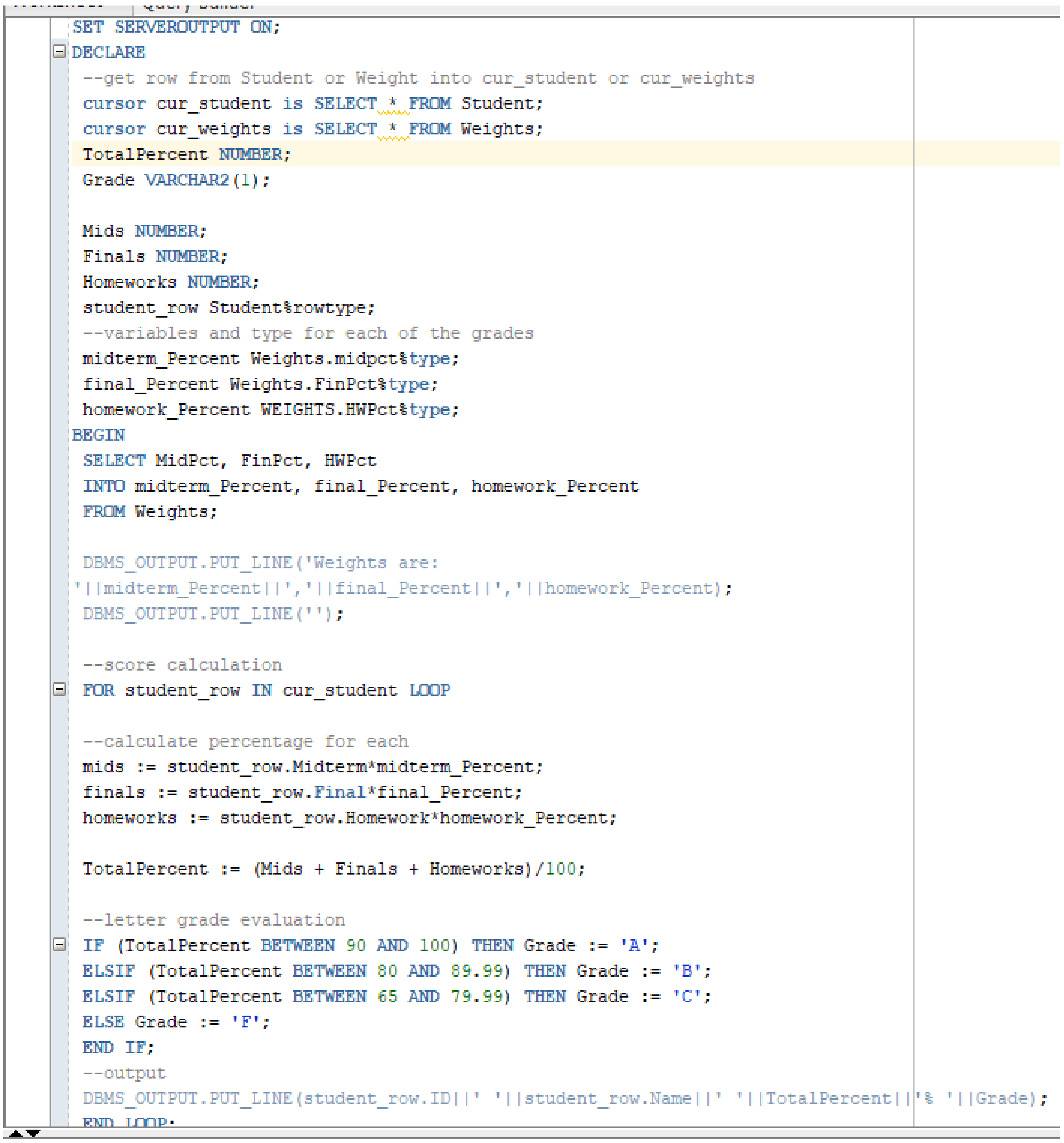
```
COMMIT;
```

<u>Write an anonymous PL/SQL block that will do the following:</u>

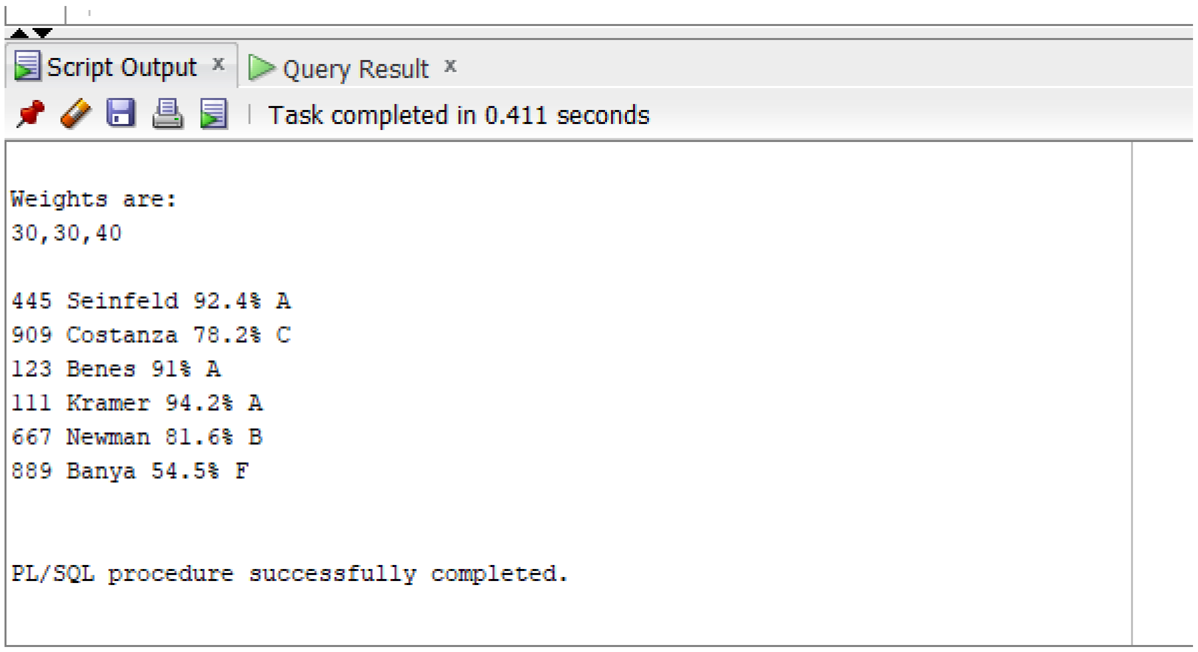
First, report the three weights found in the WEIGHTS table. (You may assume that the WEIGHTS table contains only one record.) Next, output the name of each student in the STUDENT table and their overall score, computed as x percent Midterm, y percent Final, and z percent Homework, where x, y, and z are the corresponding percentages found in the WEIGHTS table. (You may assume that x+y+z=100.) Also convert each student's overall score to a letter grade by the rule 90-100=A, 80-89.99=B, 65-79.99=C, 0-64.99=F, and include the letter grade in the output. Output each student's information on a separate line. For the sample data given above, the output should be:

```
Weights are 30, 30, 40
445 Seinfeld 92.1 A
909 Costanza 78.2 C
123 Benes 91 A
111 Kramer 94.2 A
667 Newman 81.2 B
889 Banya 54.5 F
```

Of course, this is just an example – your PL/SQL block should work in general, not just for the given sample data.



```
SET SERVEROUTPUT ON;
DECLARE
 --get row from Student or Weight into cur_student or cur_weights
 cursor cur_student is SELECT * FROM Student;
 cursor cur_weights is SELECT * FROM Weights;
 TotalPercent NUMBER;
 Grade VARCHAR2(1);

 Mids NUMBER;
 Finals NUMBER;
 Homeworks NUMBER;
 student_row Student%rowtype;
 --variables and type for each of the grades
 midterm_Percent Weights.midpct%type;
 final_Percent Weights.FinPct%type;
 homework_Percent WEIGHTS.HWPct%type;
BEGIN
 SELECT MidPct, FinPct, HWPct
 INTO midterm_Percent, final_Percent, homework_Percent
 FROM Weights;

 DBMS_OUTPUT.PUT_LINE('Weights are:
'||midterm_Percent||','||final_Percent||','||homework_Percent);
 DBMS_OUTPUT.PUT_LINE('');

 --score calculation
 FOR student_row IN cur_student LOOP

 --calculate percentage for each
 mids := student_row.Midterm*midterm_Percent;
 finals := student_row.Final*final_Percent;
 homeworks := student_row.Homework*homework_Percent;

 TotalPercent := (Mids + Finals + Homeworks)/100;

 --letter grade evaluation
 IF (TotalPercent BETWEEN 90 AND 100) THEN Grade := 'A';
 ELSIF (TotalPercent BETWEEN 80 AND 89.99) THEN Grade := 'B';
 ELSIF (TotalPercent BETWEEN 65 AND 79.99) THEN Grade := 'C';
 ELSE Grade := 'F';
 END IF;
 --output
 DBMS_OUTPUT.PUT_LINE(student_row.ID||' '||student_row.Name||' '||TotalPercent||'% '||Grade);
 END LOOP;
```



```
Weights are:
30,30,40

445 Seinfeld 92.4% A
909 Costanza 78.2% C
123 Benes 91% A
111 Kramer 94.2% A
667 Newman 81.6% B
889 Banya 54.5% F


PL/SQL procedure successfully completed.
```

Part 3

Consider the SECTION and ENROLLMENT tables defined by the following script, which also populates the SECTION table;

```
DROP TABLE ENROLLMENT CASCADE CONSTRAINTS;
DROP TABLE SECTION CASCADE CONSTRAINTS;

CREATE TABLE SECTION(
 SectionID     CHAR(5),
 Course   VARCHAR2(7),
 Students NUMBER DEFAULT 0,
 CONSTRAINT PK_SECTION
          PRIMARY KEY (SectionID)
);

CREATE TABLE ENROLLMENT(
 SectionID     CHAR(5),
 StudentID     CHAR(7),
 CONSTRAINT PK_ENROLLMENT
          PRIMARY KEY (SectionID, StudentID),
 CONSTRAINT FK_ENROLLMENT_SECTION
```

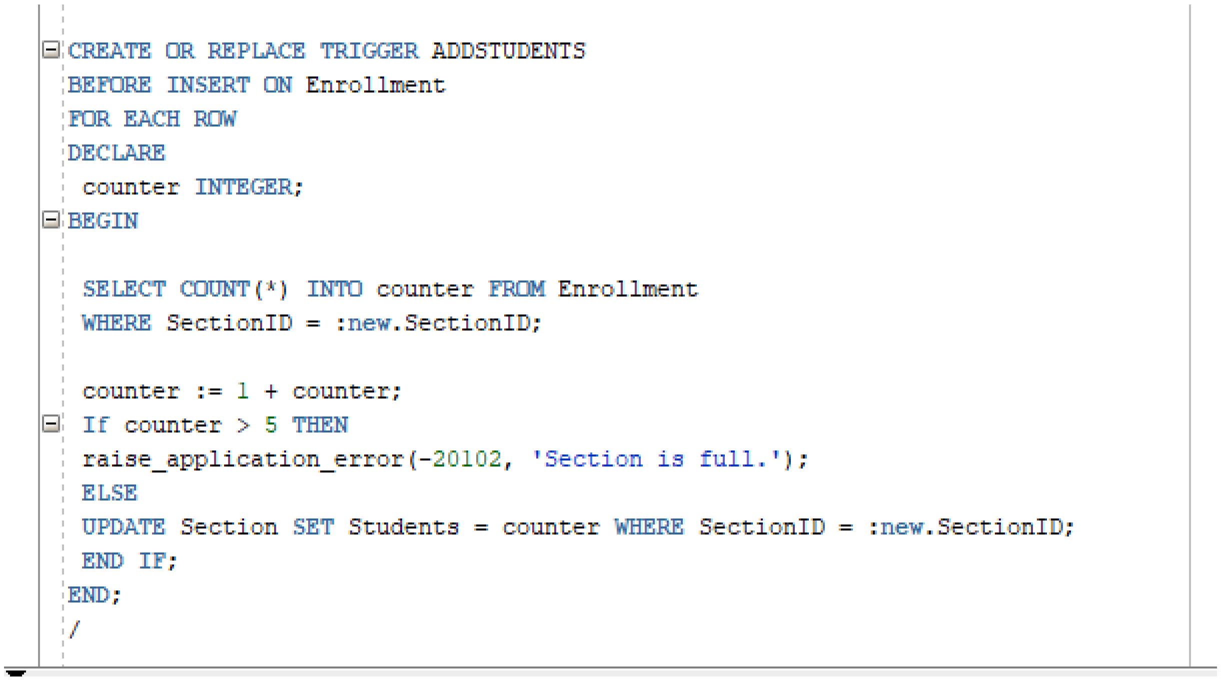
```
          FOREIGN KEY (SectionID)
          REFERENCES SECTION (SectionID)
);

INSERT INTO SECTION (SectionID, Course) VALUES ( '12345', 'CSC 355'
);
INSERT INTO SECTION (SectionID, Course) VALUES ( '22109', 'CSC 309'
);
INSERT INTO SECTION (SectionID, Course) VALUES ( '99113', 'CSC 300'
);
INSERT INTO SECTION (SectionID, Course) VALUES ( '99114', 'CSC 300'
);
COMMIT;
SELECT * FROM SECTION;
```

The Students attribute of SECTION should store a count of how many students are enrolled in the section – that is, the number of records in ENROLLMENT with that SectionID – and its value should never exceed five (they are very small sections…). Your task is to write two triggers that will maintain the value of the Students attribute as changes are made to the ENROLLMENT table.

Write definitions of the following two triggers:

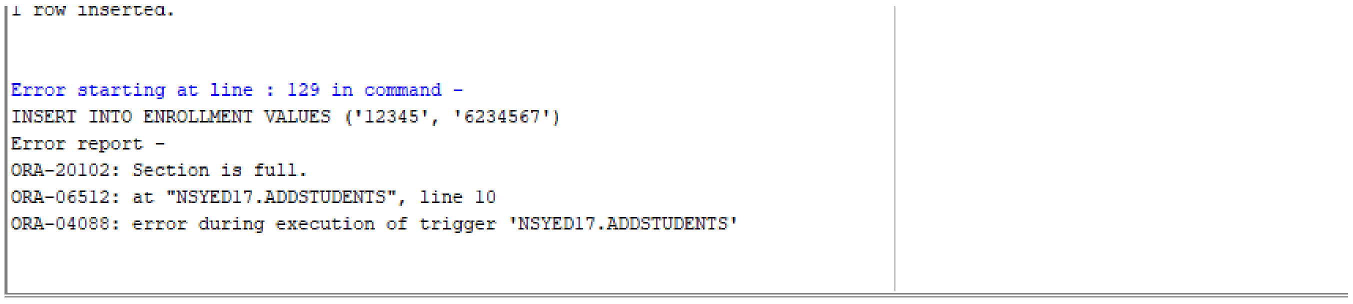
A. Write a trigger that will fire when a user attempts to INSERT a row into ENROLLMENT. This trigger will check the value of SECTION.Students for the corresponding section. If SECTION.Students is less than 5, then there is still room in the section so allow the insert and update SECTION.Students. If SECTION.Students is equal to 5, then the section is full so it will cancel the INSERT and display an error message stating that the section is full.
You can raise an error using:
raise_application_error(-20102, '[Place your error message here]');

Sample Data:
INSERT INTO ENROLLMENT VALUES ('12345', '1234567');
INSERT INTO ENROLLMENT VALUES ('12345', '2234567');
INSERT INTO ENROLLMENT VALUES ('12345', '3234567');
INSERT INTO ENROLLMENT VALUES ('12345', '4234567');
INSERT INTO ENROLLMENT VALUES ('12345', '5234567');
INSERT INTO ENROLLMENT VALUES ('12345', '6234567');
SELECT * FROM Section;
SELECT * FROM Enrollment;



```
CREATE OR REPLACE TRIGGER ADDSTUDENTS
BEFORE INSERT ON Enrollment
FOR EACH ROW
DECLARE
 counter INTEGER;
BEGIN

 SELECT COUNT(*) INTO counter FROM Enrollment
 WHERE SectionID = :new.SectionID;

 counter := 1 + counter;
 If counter > 5 THEN
 raise_application_error(-20102, 'Section is full.');
 ELSE
 UPDATE Section SET Students = counter WHERE SectionID = :new.SectionID;
 END IF;
END;
/
```

The last insert should return an error message that looks like:
Error starting at line : 27 in command -
INSERT INTO ENROLLMENT VALUES ('12345', '6234567')
Error report -
SQL Error: ORA-20200: Section is full.
ORA-06512: at "ARASIN.ADDSTUDENT", line 14
ORA-04088: error during execution of trigger 'ARASIN.ADDSTUDENT'

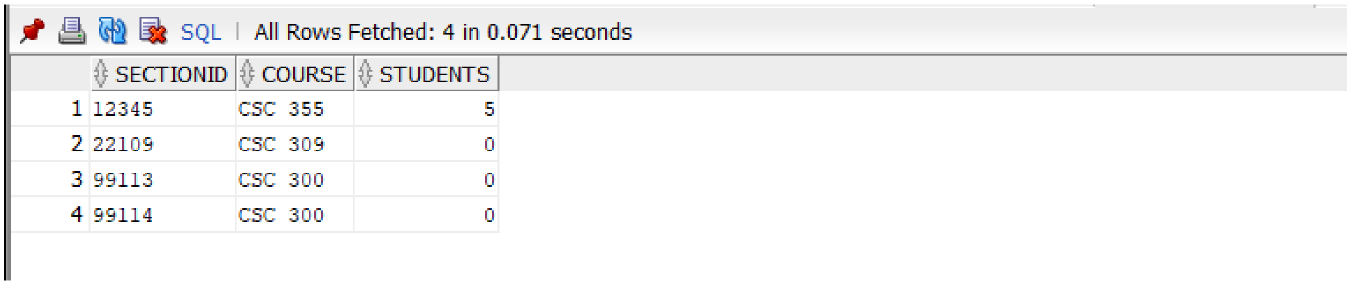


```
1 row inserted.


Error starting at line : 129 in command -
INSERT INTO ENROLLMENT VALUES ('12345', '6234567')
Error report -
ORA-20102: Section is full.
ORA-06512: at "NSYED17.ADDSTUDENTS", line 10
ORA-04088: error during execution of trigger 'NSYED17.ADDSTUDENTS'
```

The output from the SELECT queries should look like:

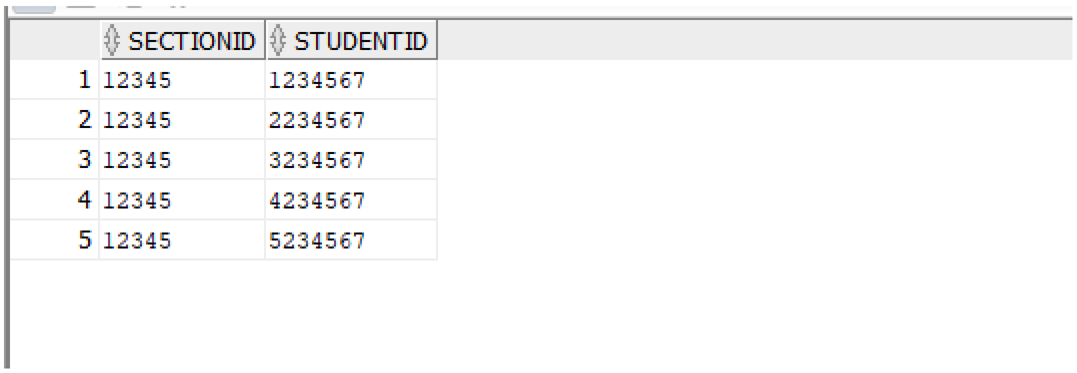
```
SECTIONID COURSE    STUDENTS
--------- ------- ----------
12345     CSC 355        5
22109     CSC 309        0
99113     CSC 300        0
99114     CSC 300        0
```

```
SECTIONID STUDENTID
--------- ---------
12345     1234567
12345     2234567
12345     3234567
12345     4234567
12345     5234567
```



SQL | All Rows Fetched: 4 in 0.071 seconds

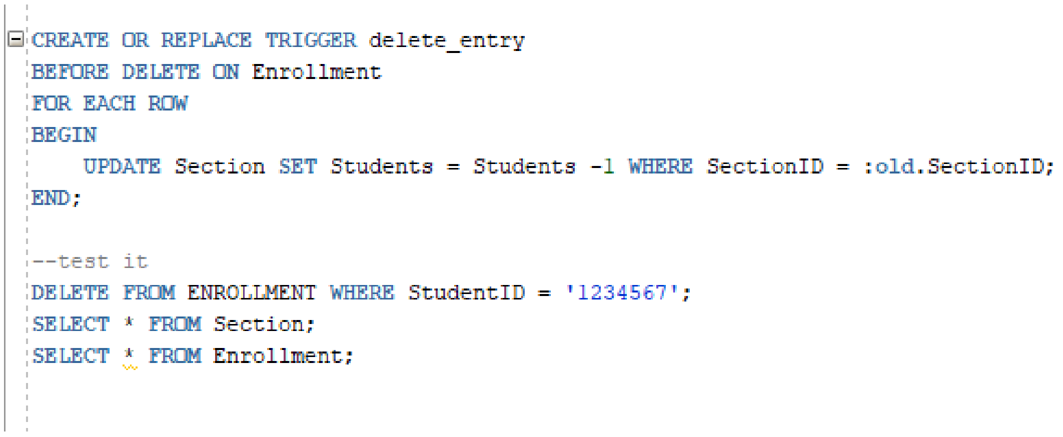
	SECTIONID	COURSE	STUDENTS
1	12345	CSC 355	5
2	22109	CSC 309	0
3	99113	CSC 300	0
4	99114	CSC 300	0



	SECTIONID	STUDENTID
1	12345	1234567
2	12345	2234567
3	12345	3234567
4	12345	4234567
5	12345	5234567

<u>B. Write a trigger</u> that will fire when a user attempts to DELETE one or more rows from ENROLLMENT. This trigger will update the values of SECTION.Students for any affected sections to make sure they are accurate after the rows are deleted, by decreasing the value of SECTION.Students by one each time a student is removed from a section.

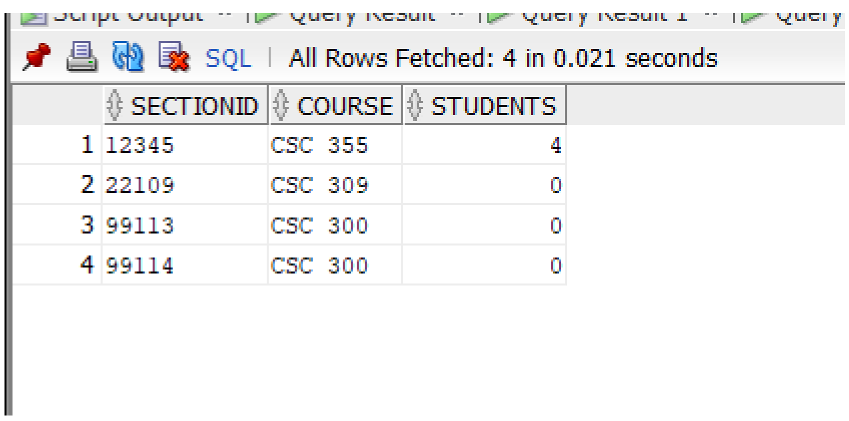
Sample Data:
DELETE FROM ENROLLMENT WHERE StudentID = '1234567';
SELECT * FROM Section;
SELECT * FROM Enrollment;



```
CREATE OR REPLACE TRIGGER delete_entry
BEFORE DELETE ON Enrollment
FOR EACH ROW
BEGIN
    UPDATE Section SET Students = Students -1 WHERE SectionID = :old.SectionID;
END;

--test it
DELETE FROM ENROLLMENT WHERE StudentID = '1234567';
SELECT * FROM Section;
SELECT * FROM Enrollment;
```

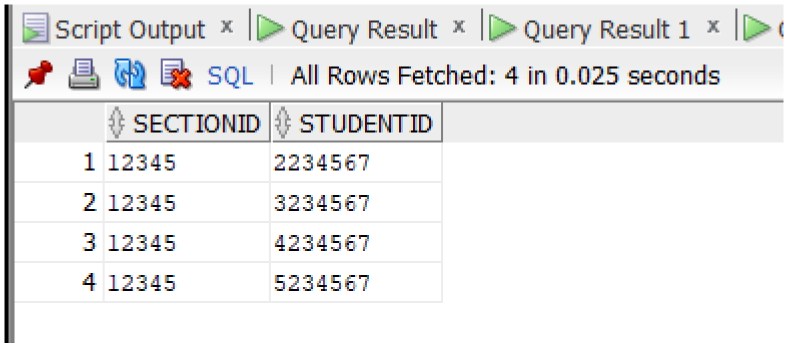
The output from the SELECT queries should look like:

```
SECTIONID COURSE    STUDENTS
--------- ------- ----------
12345     CSC 355        4
22109     CSC 309        0
99113     CSC 300        0
99114     CSC 300        0

SECTIONID STUDENTID
--------- ---------
12345     2234567
12345     3234567
12345     4234567
12345     5234567
```



All Rows Fetched: 4 in 0.021 seconds

	SECTIONID	COURSE	STUDENTS
1	12345	CSC 355	4
2	22109	CSC 309	0
3	99113	CSC 300	0
4	99114	CSC 300	0



All Rows Fetched: 4 in 0.025 seconds

	SECTIONID	STUDENTID
1	12345	2234567
2	12345	3234567
3	12345	4234567
4	12345	5234567