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.

```
import pandas as pd
import numpy

def employeesPandas():
   randn = numpy.random.randn

   names = ['FirstName', 'MiddleInitial', 'LastName', 'SSN', 'DOB', 'Address', 'City', 'State',
'Sex', 'Pay', 'SupervisorSSN', 'DeptId']

   employees = pd.read_csv("Employee.txt", names=names)

   df = pd.DataFrame(data = employees)
```

a. Find all male employees

```
males = df[df['Sex'] == 'M']
   print('All male employees:')
   print(males)
   print('-'*30)
```

b. Find the highest salary for female employees

```
highFemale = df[df['Sex'] == 'F']['Pay'].max()
   print('Highest female salary:')
   print(highFemale)
   print('-'*30)
```

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

```
dg = df['Pay'].groupby(df['MiddleInitial'])
   for gname, gvalue in dg:
      print('Middle initial: ' + gname)
      print(gvalue)
      print('\n')
   print('-'*30)
```

```
All male employees:
  FirstName MiddleInitial LastName        SSN         DOB       Address      City State Sex    Pay SupervisorSSN  DeptId
0     James             E     Borg  888665555  1937-11-10     450 Stone   Houston    TX   M  55000          NULL       1
2  Franklin             T     Wong  333445555  1955-12-08      638 Voss   Houston    TX   M  40000     888665555       5
3      John             B    Smith  123456789  1965-01-09    731 Fondren   Houston    TX   M  30000     333445555       5
5    Ramesh             K  Narayan  666884444  1920-09-15   975 Fire Oak    Humble    TX   M  38000     333445555       5
7     Ahmad             T   Jabbar  987987987  1969-03-29     980 Dallas   Houston    TX   M  22000     987654321       4
------------------------------
Highest female salary:
37000
------------------------------
Salaries for each middle initial
Middle initial: B
3    30000
Name: Pay, dtype: int64


Middle initial: E
0    55000
8    27500
Name: Pay, dtype: int64


Middle initial: J
4    25000
Name: Pay, dtype: int64


Middle initial: K
5    38000
Name: Pay, dtype: int64


Middle initial: S
1    37000
6    25000
Name: Pay, dtype: int64


Middle initial: T
2    40000
7    22000
Name: Pay, dtype: int64
```

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', 77, 82, 84 );
INSERT INTO STUDENT VALUES ( '889', 'Banya', 52, 66, 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;
```

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

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 around:

```
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.

```
DECLARE
   midterm_weight number;
   final_weight number;
   homework_weight number;
   grade number;
   letter varchar(1);
   CURSOR st_cursor IS (SELECT ID, Name, Midterm, Final, Homework  FROM STUDENT);
   st_id STUDENT.ID%TYPE;
   st_name STUDENT.Name%TYPE;
   st_midterm STUDENT.Midterm%TYPE;
   st_final STUDENT.Final%TYPE;
   st_Homework STUDENT.Homework%TYPE;
BEGIN
   SELECT MidPct INTO midterm_weight FROM WEIGHTS;
   SELECT FinPct INTO final_weight FROM WEIGHTS;
   SELECT HWPct INTO homework_weight FROM WEIGHTS;
   dbms_output.put_line('Weights are ' || midterm_weight || ', ' || final_weight || ', ' ||
homework_weight);
   OPEN st_cursor;
   LOOP
      FETCH st_cursor INTO st_id, st_name, st_midterm, st_final, st_homework;
      EXIT WHEN st_cursor%NOTFOUND;
      grade := (st_midterm * (midterm_weight/100)) + (st_final * (final_weight/100)) + (st_homework *
(homework_weight/100));
      IF grade > 89 THEN
         letter := 'A';
      ELSIF grade > 79 THEN
         letter := 'B';
      ELSIF grade > 69 THEN
         letter := 'C';
      ELSIF grade > 59 THEN
         letter := 'D';
      ELSE
         letter := 'F';
      END IF;
      dbms_output.put_line(st_id || ' ' || st_name || ' ' || grade || ' ' || letter);
   END LOOP;
   CLOSE st_cursor;
END;
```

```
PL/SQL procedure successfully completed.

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 81.3 B
889 Banya 55.4 F
```

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;

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'

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

```
Error starting at line : 157 in command -
INSERT INTO ENROLLMENT VALUES ('12345', '6234567')
Error at Command Line : 157 Column : 13
Error report -
SQL Error: ORA-20102: Section is full.
ORA-06512: at "CHANLON1.ENROLLMENTTRIGGER", line 6
ORA-04088: error during execution of trigger 'CHANLON1.ENROLLMENTTRIGGER'
```

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

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

```
CREATE OR REPLACE TRIGGER EnrollmentTrigger
BEFORE INSERT ON ENROLLMENT
FOR EACH ROW
DECLARE
  student_ct NUMBER(3);
BEGIN
  SELECT Students INTO student_ct FROM SECTION WHERE SectionID  = :NEW.SectionID;
  IF student_ct >= 5 THEN
    raise_application_error(-20102, 'Section is full.');
  ELSE
    UPDATE SECTION
    SET Students = Students + 1
    WHERE SectionID = :NEW.SectionID;
  END IF;
END;
/
```

B. Write a trigger 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;

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
```

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

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

```
CREATE OR REPLACE TRIGGER DeleteEnrollmentTrigger
AFTER DELETE ON ENROLLMENT
FOR EACH ROW
BEGIN
   UPDATE SECTION
   SET Students = Students - 1
   WHERE SectionID = :OLD.SectionID;
END;
/
```