Name: Syed Noor Razi Ali

2070326

DSC 450: Database Processing for Large-Scale Analytics

Take-home Midterm

Do not forget to include all python code and SQL code for every question (screenshots are only required when the question specifically asked for it).

Part 1

(9 points) Please give a "True" or a "False" rating to each statement below.

a. A schema that is in Third Normal Form (3NF) must also be in Second Normal Form (2NF).
TRUE

b. A foreign key can contain a NULL.
TRUE

c. The union rule allows combining functional dependencies A→CD and A→EF into a functional dependency A→CDEF.
TRUE

d. SQL query output is always sorted by the primary key.
FALSE

e. A column that appears in the WHERE clause must always appear in the SELECT clause.
FALSE

f. A UNIQUE constraint on a column allows insertion of a NULL.
TRUE

Part 2

a. **(3 points) Write a SQL column definition for PurchaseAmt (just for one column, don't worry about the rest of the table) that ensures that the purchase amount must be between 0 and 999.99 (column definition should enforce both the number precision and greater-than-0 requirement).**

ANS.- PurchaseAmt NUMBER (5,3) CHECK (PurchaseAmt > 0)

b. **(3 points) Write a SQL column definition for Street (just for one column, don't worry about the rest of the table) that ensures that the street address is up to 18 characters and has to start from 'rue' (hint: CHECK constraint can use LIKE operator).**

ANS.- Street CHAR (18) CHECK (Street LIKE 'rue%')

c. (12 points) The following set of relations records information about university students, courses and assigned grades. The Student relation contains information about the student, including the name (full name is stored in one column), address and their year of graduation. The Course relation records information about courses: course name (primary key), course department and the number of credits provided by the course. Finally, the Grade relation records information about the grades given; CName is the foreign key referring to the primary key of the Course relation and StudentID is the foreign key referring to the primary key of the Student relation. The grades are a numeric value given on a 4-point system.

Student(StudentID, Name, Address, GradYear)

Grade(CName, StudentID, CGrade)

Course(CName, Department, Credits)

For each part below, write a single SQL query.

Q1. Display the list of student IDs and names for the students who graduated in most recent two years. You can assume that GradYear is an integer, but your query is not allowed to assume any particular year.

ANS-

```
SELECT StudentID, Name FROM Student
WHERE (SELECT MIN (GradYear) +2 FROM Student) >= GradYear;
```

Q2. Display student names and their taken course names for all students with the middle name of 'Muriel'. You may assume that name is always written as 'First Middle Last'. Your query output should be sorted by grade.

ANS-

```
SELECT s.Name, g.CName FROM Student s, Grade g
WHERE g.StudentID = s.StudentID AND s.Name LIKE '%Muriel%' ORDER BY CGrade;
```

Q3. For students who are either not enrolled in any courses or are enrolled in only 1 course, list those student's names and graduation years.

```
SELECT StudentID, NAME FROM Student
WHERE NOT EXISTS (SELECT * FROM Grade WHERE s.StudentID = g.StudentID) UNION
SELECT StudentID, NAME FROM Student
 WHERE (SELECT count(*) FROM Grade
WHERE s.SudentID = g.StudentID) = 1;
```

Q4. Update all student records, to increase the graduation year by 3 for all students who live in Chicago

ANS-

```
UPDATE Student SET GradYear = GradYear + 3
WHERE Address LIKE '%Chicago%';
```

Q5. Modify the course table to add a Chair column that can be up to 28 characters (that question requires a DDL rather than a DML SQL statement)

ANS-

```
ALTER TABLE Course
ADD (Chair VARCHAR (28));
```

Part 3

a. (8 points)

- For the table below, fill in the missing values in W column, consistent with functional dependencies: XY→ZA , A→W. You can make any necessary assumptions, but be sure to state them.

X	Y	Z	W	A
1	1	1000	Cat	2
1	2	4000	Hawk	4
1	3	3000	Dog	3
2	1	1000	Hawk	3
2	2	2000	Dog	4
2	3	5000	Null	5

- Given the schema R and the following functional dependencies:

R(X, Y, Z, W, A) with XY$\rightarrow$ZA , A$\rightarrow$W
does XY determine W? (XY$\rightarrow$W?) Why or why not?

Yes; Since XY -> ZA, then XY -> A, and A->W, hence XY->A; XY determines W by transitive dependency.

- Suppose that you were also given relation S:
  S(P, Q, U, F, N)
  What functional dependencies (if any) can you assume?

PUN -> QF; PU -> QF; PU -> Q; PU -> F; PN -> QF; PN -> Q; PN -> F; PUN -> Q; PUN -> F; P -> Q; P ->F; U -> Q; U -> F; N -> Q; N -> F; UN -> Q; UN -> F

b. (8 points) Consider a TVShows table that keeps track of different TV shows. The table stores the show name and the year to which the entry refers. Additionally, each row stores channel name, length of the show, the average cost of an episode, and the filming location's zip, city and state. Moreover, each entry contains the name of the lead actor and their salary.

The table is already in First Normal Form, and its primary key is (ShowTitle, Year).

The schema for the TVShows table is:

(ShowTitle, Year, Channel, Length, Cost, Zip, City, State, Lead, Salary)

You are given the following functional dependencies:

ShowTitle $\rightarrow$ Cost

Zip $\rightarrow$ City, State

Lead $\rightarrow$ Salary

ShowTitle, Year $\rightarrow$ Channel, Cost, Length

- Remove any existing partial dependencies to create a set of linked relational schema (copying functional dependencies does not define a schema, be sure to include primary/foreign keys here) in Second Normal Form.

Table1(ShowTitle, Cost)

$\downarrow$

Table2(ShowTitle, Year, Channel, Length, Cost, Zip, City, State, Lead, Salary)

- Remove any existing transitive dependencies to create a set of linked relational schemas in Third Normal Form.

$$\begin{tikzcd}
\text{(ShowTitle, Cost, Length)} \arrow[dr] & \text{(Salary, Lead)} \arrow[d] & \text{(Zip, City, State)} \arrow[dl] \\
& \text{(ShowTitle, Year, Channel, Lead, Zip)} &
\end{tikzcd}$$

c. (7 points)

Given the schema R and the following functional dependencies:

R(A, B, C, D, E) with AB→C, C→D

- Describe how to identify a primary key for relation R (the primary key is a minimal set of columns that determines all columns in the relation such that **?** → ABCDE)

AB -> C, and C -> D, then AB -> D is true; since E is left out, we can make it a primary key too.

So, ABE -> ABCDE

- Decompose relation R into a relational schema in third normal form

$$\begin{tikzcd}
\text{(A, B, E)} \arrow[dr, shift left] \arrow[dr, shift right] & & \text{(C, D)} \arrow[dl] \\
& \text{(A, B, C)} &
\end{tikzcd}$$

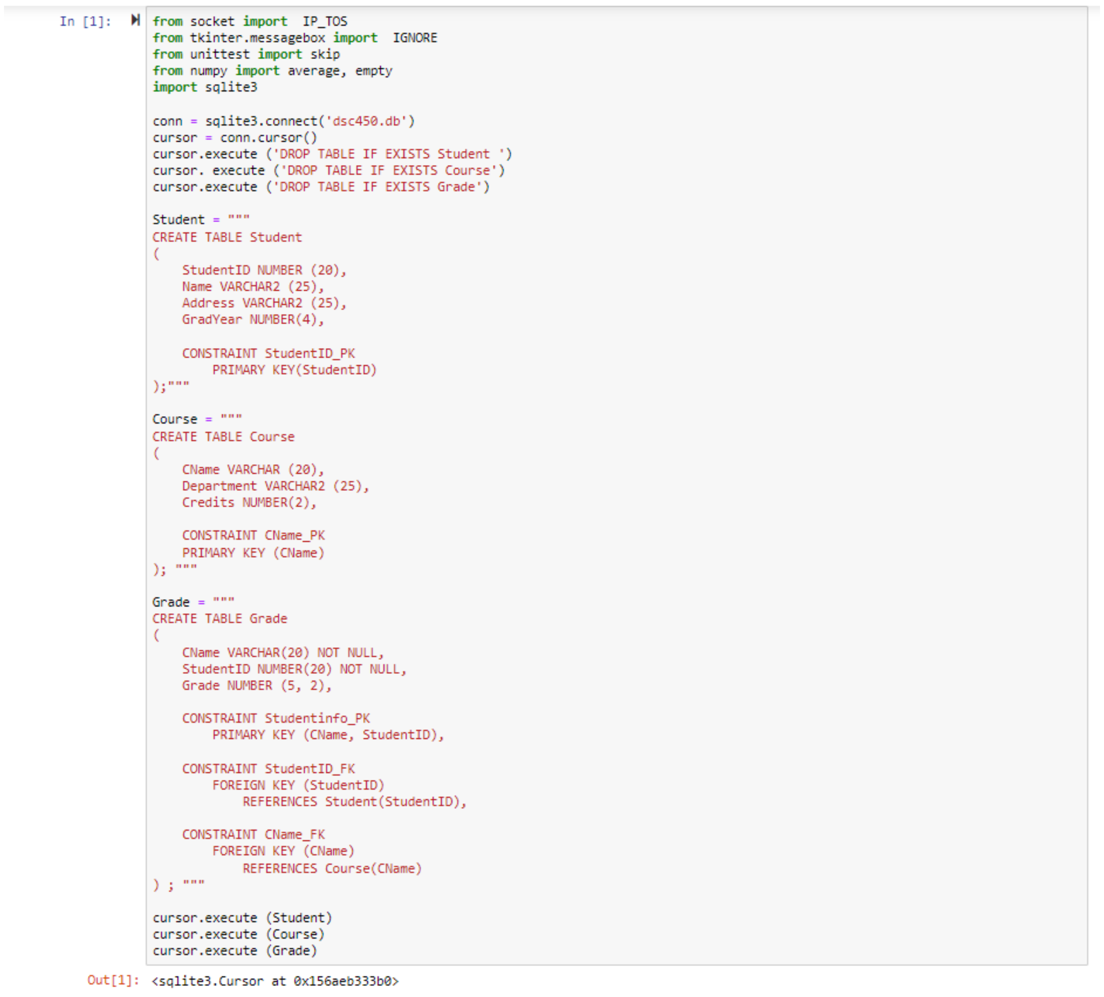
Part 4

(50 points)

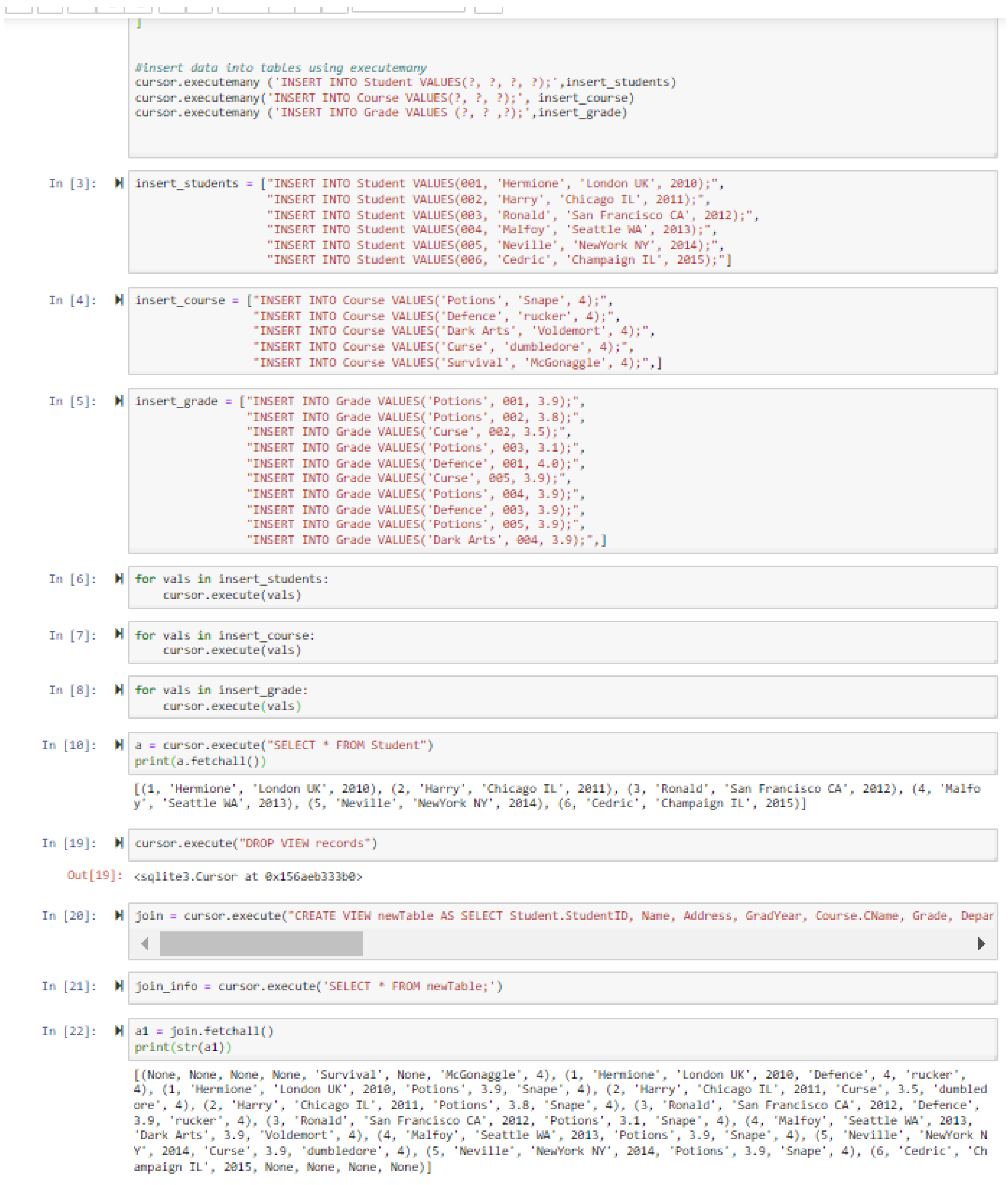
Create the schema from Part 2-c in SQLite and populate it with data of at least 5 students, 4 courses, and 7 enrollments (at least one of the students should not be enrolled in any courses and at least one course should have zero current enrollments).

You can submit code in this document or as a separate .py file. If you submit it separately, please clearly document which part each answer matches. Do not submit each part in a separate .py file.

a) Create a view that joins the three tables, including all of the records from student table (i.e., including the non-enrolled students).



```
In [1]:
from socket import  IP_TOS
from tkinter.messagebox import  IGNORE
from unittest import skip
from numpy import average, empty
import sqlite3

conn = sqlite3.connect('dsc450.db')
cursor = conn.cursor()
cursor.execute ('DROP TABLE IF EXISTS Student ')
cursor. execute ('DROP TABLE IF EXISTS Course')
cursor.execute ('DROP TABLE IF EXISTS Grade')

Student = """
CREATE TABLE Student
(
    StudentID NUMBER (20),
    Name VARCHAR2 (25),
    Address VARCHAR2 (25),
    GradYear NUMBER(4),

    CONSTRAINT StudentID_PK
        PRIMARY KEY(StudentID)
);"""

Course = """
CREATE TABLE Course
(
    CName VARCHAR (20),
    Department VARCHAR2 (25),
    Credits NUMBER(2),

    CONSTRAINT CName_PK
    PRIMARY KEY (CName)
); """

Grade = """
CREATE TABLE Grade
(
    CName VARCHAR(20) NOT NULL,
    StudentID NUMBER(20) NOT NULL,
    Grade NUMBER (5, 2),

    CONSTRAINT Studentinfo_PK
        PRIMARY KEY (CName, StudentID),

    CONSTRAINT StudentID_FK
        FOREIGN KEY (StudentID)
            REFERENCES Student(StudentID),

    CONSTRAINT CName_FK
        FOREIGN KEY (CName)
            REFERENCES Course(CName)
) ; """

cursor.execute (Student)
cursor.execute (Course)
cursor.execute (Grade)

Out[1]: <sqlite3.Cursor at 0x156aeb333b0>
```



```
    ]

    #insert data into tables using executemany
    cursor.executemany ('INSERT INTO Student VALUES(?, ?, ?, ?);',insert_students)
    cursor.executemany('INSERT INTO Course VALUES(?, ?, ?);', insert_course)
    cursor.executemany ('INSERT INTO Grade VALUES (?, ? ,?);',insert_grade)
```

```
In [3]:  insert_students = ["INSERT INTO Student VALUES(001, 'Hermione', 'London UK', 2010);",
                            "INSERT INTO Student VALUES(002, 'Harry', 'Chicago IL', 2011);",
                            "INSERT INTO Student VALUES(003, 'Ronald', 'San Francisco CA', 2012);",
                            "INSERT INTO Student VALUES(004, 'Malfoy', 'Seattle WA', 2013);",
                            "INSERT INTO Student VALUES(005, 'Neville', 'NewYork NY', 2014);",
                            "INSERT INTO Student VALUES(006, 'Cedric', 'Champaign IL', 2015);"]
```

```
In [4]:  insert_course = ["INSERT INTO Course VALUES('Potions', 'Snape', 4);",
                          "INSERT INTO Course VALUES('Defence', 'rucker', 4);",
                          "INSERT INTO Course VALUES('Dark Arts', 'Voldemort', 4);",
                          "INSERT INTO Course VALUES('Curse', 'dumbledore', 4);",
                          "INSERT INTO Course VALUES('Survival', 'McGonaggle', 4);",]
```

```
In [5]:  insert_grade = ["INSERT INTO Grade VALUES('Potions', 001, 3.9);",
                         "INSERT INTO Grade VALUES('Potions', 002, 3.8);",
                         "INSERT INTO Grade VALUES('Curse', 002, 3.5);",
                         "INSERT INTO Grade VALUES('Potions', 003, 3.1);",
                         "INSERT INTO Grade VALUES('Defence', 001, 4.0);",
                         "INSERT INTO Grade VALUES('Curse', 005, 3.9);",
                         "INSERT INTO Grade VALUES('Potions', 004, 3.9);",
                         "INSERT INTO Grade VALUES('Defence', 003, 3.9);",
                         "INSERT INTO Grade VALUES('Potions', 005, 3.9);",
                         "INSERT INTO Grade VALUES('Dark Arts', 004, 3.9);",]
```

```
In [6]:  for vals in insert_students:
             cursor.execute(vals)
```

```
In [7]:  for vals in insert_course:
             cursor.execute(vals)
```

```
In [8]:  for vals in insert_grade:
             cursor.execute(vals)
```

```
In [10]: a = cursor.execute("SELECT * FROM Student")
         print(a.fetchall())

         [(1, 'Hermione', 'London UK', 2010), (2, 'Harry', 'Chicago IL', 2011), (3, 'Ronald', 'San Francisco CA', 2012), (4, 'Malfo
         y', 'Seattle WA', 2013), (5, 'Neville', 'NewYork NY', 2014), (6, 'Cedric', 'Champaign IL', 2015)]
```

```
In [19]: cursor.execute("DROP VIEW records")

Out[19]: <sqlite3.Cursor at 0x156aeb333b0>
```

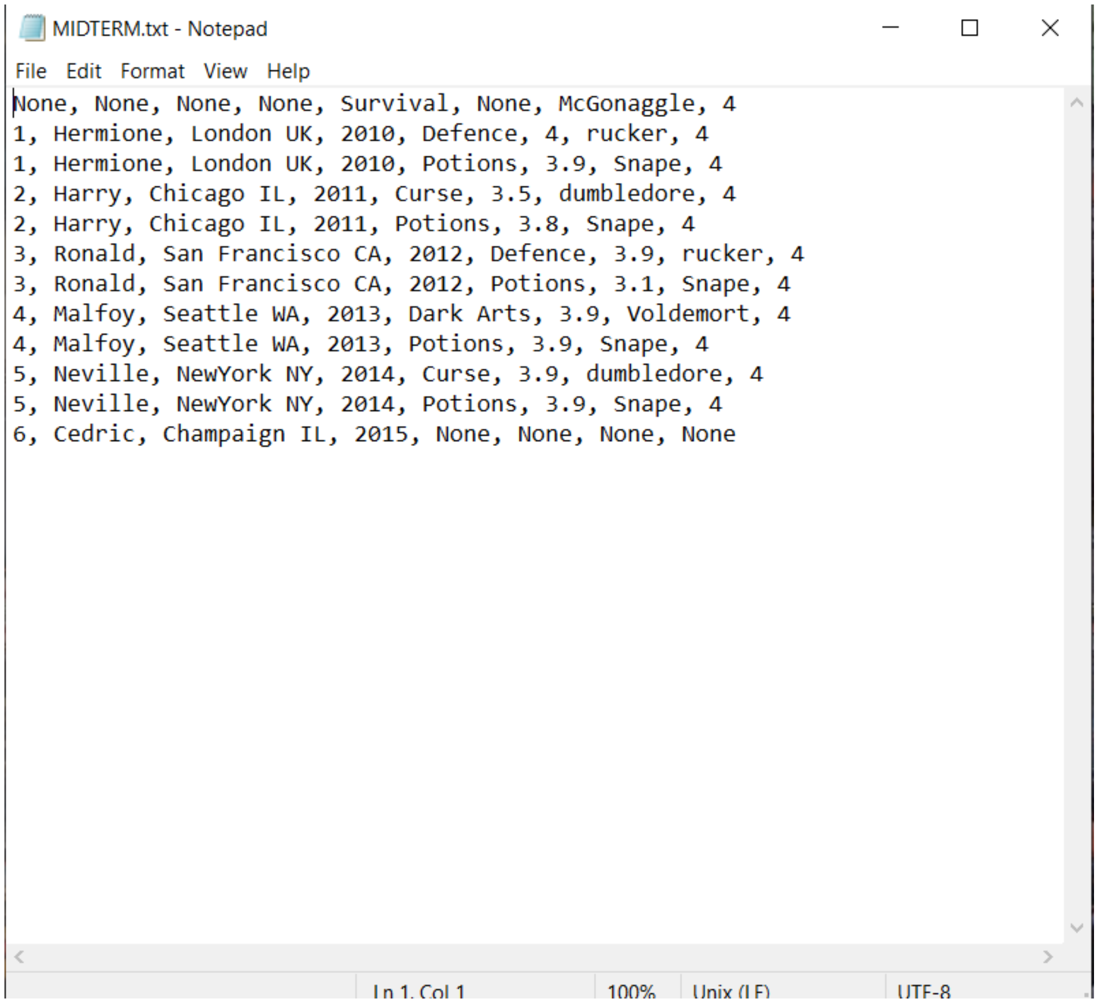
```
In [20]: join = cursor.execute("CREATE VIEW newTable AS SELECT Student.StudentID, Name, Address, GradYear, Course.CName, Grade, Depar
```

```
In [21]: join_info = cursor.execute('SELECT * FROM newTable;')
```

```
In [22]: a1 = join.fetchall()
         print(str(a1))

         [(None, None, None, None, 'Survival', None, 'McGonaggle', 4), (1, 'Hermione', 'London UK', 2010, 'Defence', 4, 'rucker',
         4), (1, 'Hermione', 'London UK', 2010, 'Potions', 3.9, 'Snape', 4), (2, 'Harry', 'Chicago IL', 2011, 'Curse', 3.5, 'dumbled
         ore', 4), (2, 'Harry', 'Chicago IL', 2011, 'Potions', 3.8, 'Snape', 4), (3, 'Ronald', 'San Francisco CA', 2012, 'Defence',
         3.9, 'rucker', 4), (3, 'Ronald', 'San Francisco CA', 2012, 'Potions', 3.1, 'Snape', 4), (4, 'Malfoy', 'Seattle WA', 2013,
         'Dark Arts', 3.9, 'Voldemort', 4), (4, 'Malfoy', 'Seattle WA', 2013, 'Potions', 3.9, 'Snape', 4), (5, 'Neville', 'NewYork N
         Y', 2014, 'Curse', 3.9, 'dumbledore', 4), (5, 'Neville', 'NewYork NY', 2014, 'Potions', 3.9, 'Snape', 4), (6, 'Cedric', 'Ch
         ampaign IL', 2015, None, None, None, None)]
```

b) Write and execute python code that uses that view to export all data into a single .txt file (that is a "de-normalized" 1NF file with some redundancy present). This code should include NULLs due to the non-enrolled students.

Include a screenshot of the output .txt file (in addition to the python code)

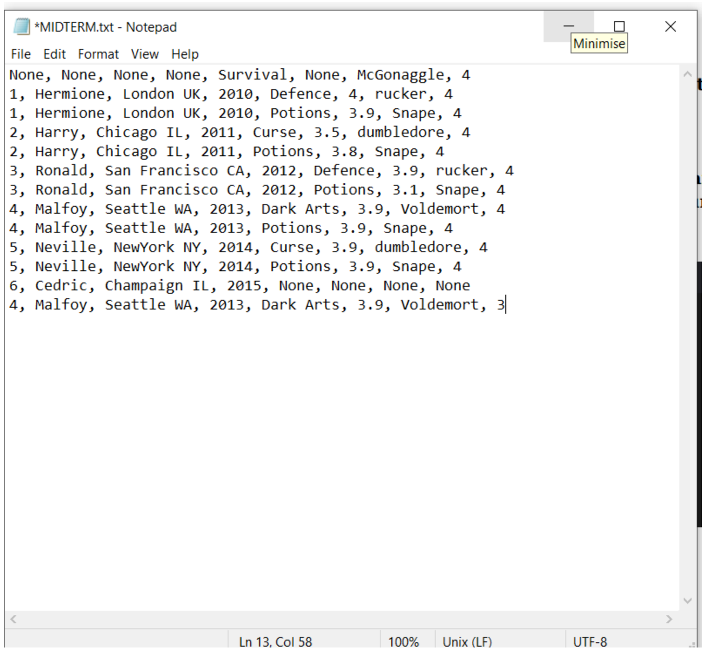


```
None, None, None, None, Survival, None, McGonaggle, 4
1, Hermione, London UK, 2010, Defence, 4, rucker, 4
1, Hermione, London UK, 2010, Potions, 3.9, Snape, 4
2, Harry, Chicago IL, 2011, Curse, 3.5, dumbledore, 4
2, Harry, Chicago IL, 2011, Potions, 3.8, Snape, 4
3, Ronald, San Francisco CA, 2012, Defence, 3.9, rucker, 4
3, Ronald, San Francisco CA, 2012, Potions, 3.1, Snape, 4
4, Malfoy, Seattle WA, 2013, Dark Arts, 3.9, Voldemort, 4
4, Malfoy, Seattle WA, 2013, Potions, 3.9, Snape, 4
5, Neville, NewYork NY, 2014, Curse, 3.9, dumbledore, 4
5, Neville, NewYork NY, 2014, Potions, 3.9, Snape, 4
6, Cedric, Champaign IL, 2015, None, None, None, None
```

c) Add a new row to the de-normalized .txt file (you can manually edit the .txt file from part b) that violates the following functional dependency:

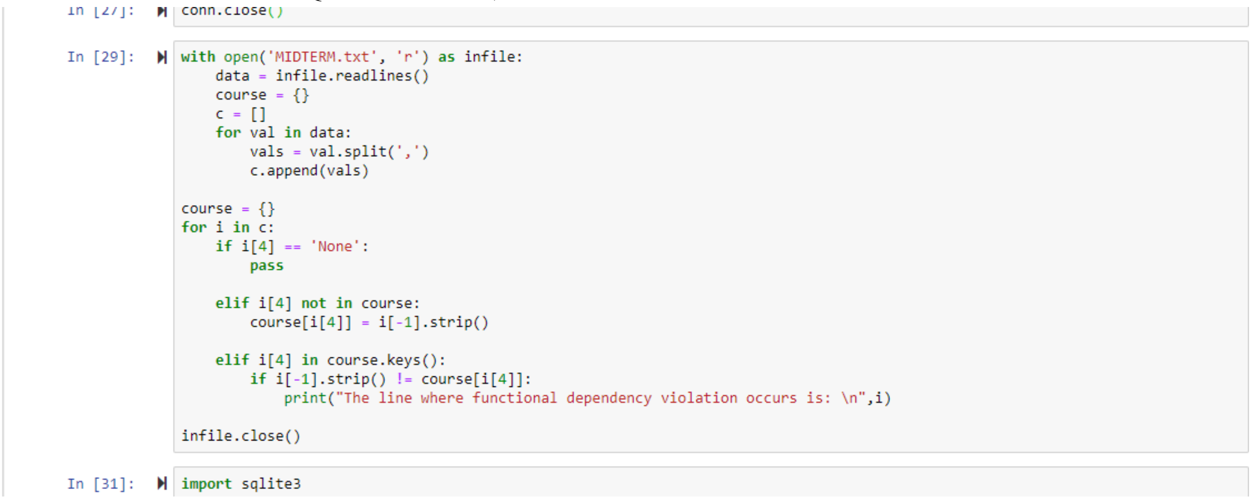
CName → Credits

(you can do so by creating a new record that repeats the course name but does not repeat the number of credits associated with this course name)



```
None, None, None, None, Survival, None, McGonaggle, 4
1, Hermione, London UK, 2010, Defence, 4, rucker, 4
1, Hermione, London UK, 2010, Potions, 3.9, Snape, 4
2, Harry, Chicago IL, 2011, Curse, 3.5, dumbledore, 4
2, Harry, Chicago IL, 2011, Potions, 3.8, Snape, 4
3, Ronald, San Francisco CA, 2012, Defence, 3.9, rucker, 4
3, Ronald, San Francisco CA, 2012, Potions, 3.1, Snape, 4
4, Malfoy, Seattle WA, 2013, Dark Arts, 3.9, Voldemort, 4
4, Malfoy, Seattle WA, 2013, Potions, 3.9, Snape, 4
5, Neville, NewYork NY, 2014, Curse, 3.9, dumbledore, 4
5, Neville, NewYork NY, 2014, Potions, 3.9, Snape, 4
6, Cedric, Champaign IL, 2015, None, None, None, None
4, Malfoy, Seattle WA, 2013, Dark Arts, 3.9, Voldemort, 3
```

d) Write python code that will identify the values for which functional dependency was violated in your .txt file (hint: when the functional dependency is valid, there is only one unique value of Credits for each CName). Your solution should detect <u>any</u> violation of the CName → Credits functional dependency, not just your example. Keep in mind that functional dependency is violated only in the pre-joined .txt file, not in the SQLite database, so this solution must read data from .txt file.



```
In [27]:  conn.close()
```

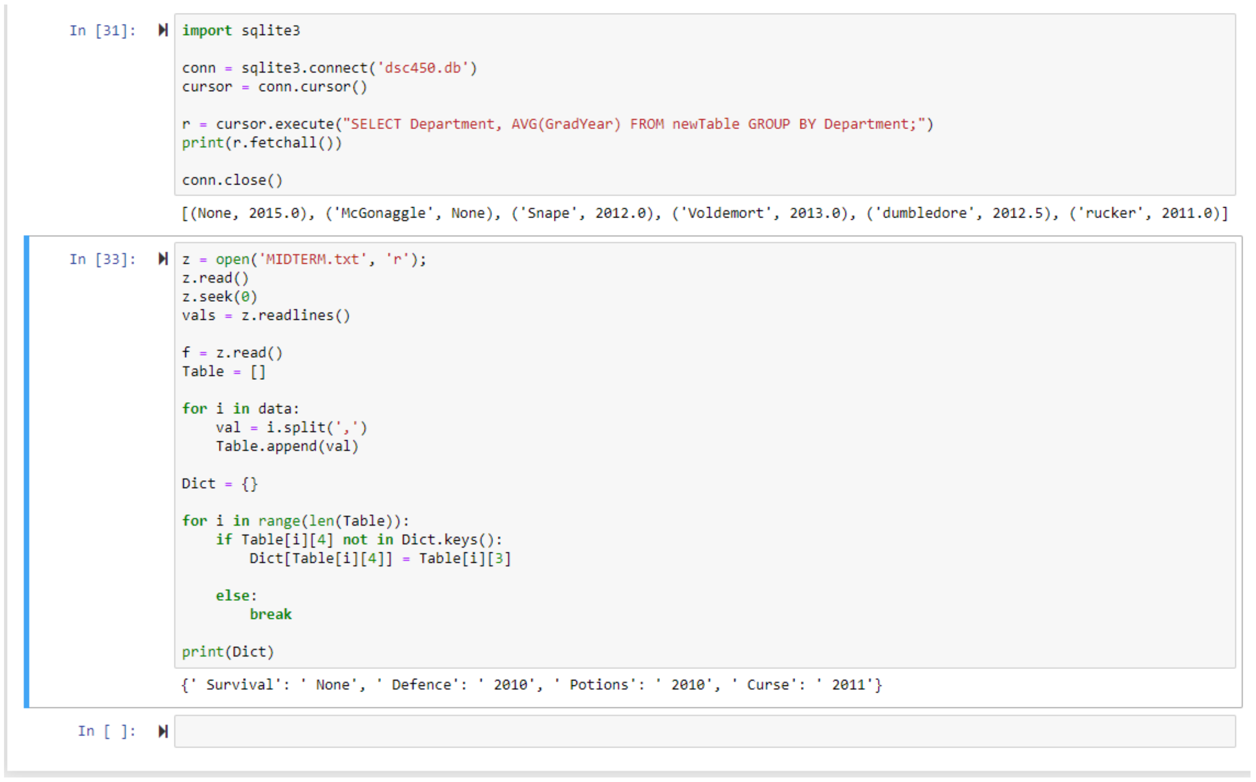
```
In [29]:  with open('MIDTERM.txt', 'r') as infile:
              data = infile.readlines()
              course = {}
              c = []
              for val in data:
                  vals = val.split(',')
                  c.append(vals)

          course = {}
          for i in c:
              if i[4] == 'None':
                  pass

              elif i[4] not in course:
                  course[i[4]] = i[-1].strip()

              elif i[4] in course.keys():
                  if i[-1].strip() != course[i[4]]:
                      print("The line where functional dependency violation occurs is: \n",i)

          infile.close()
```

```
In [31]:  import sqlite3
```

e) Suppose I have a new query: Q_e: For every department, display the average graduation year.

1. Use the view from Part 4-a to re-write query Q_e (i.e., replace the tables in the query's FROM clause by the view and rewrite the rest of the query accordingly to produce an answer).

2. Use your .txt file containing de-normalized data from part-b to answer Q_e with python instead of SQL. Note that this solution should not use SQLite or SQL, just python.



In [31]:
```
import sqlite3

conn = sqlite3.connect('dsc450.db')
cursor = conn.cursor()

r = cursor.execute("SELECT Department, AVG(GradYear) FROM newTable GROUP BY Department;")
print(r.fetchall())

conn.close()
```

```
[(None, 2015.0), ('McGonaggle', None), ('Snape', 2012.0), ('Voldemort', 2013.0), ('dumbledore', 2012.5), ('rucker', 2011.0)]
```

In [33]:
```
z = open('MIDTERM.txt', 'r');
z.read()
z.seek(0)
vals = z.readlines()

f = z.read()
Table = []

for i in data:
    val = i.split(',')
    Table.append(val)

Dict = {}

for i in range(len(Table)):
    if Table[i][4] not in Dict.keys():
        Dict[Table[i][4]] = Table[i][3]

    else:
        break

print(Dict)
```

```
{' Survival': ' None', ' Defence': ' 2010', ' Potions': ' 2010', ' Curse': ' 2011'}
```

In [ ]: