Assignment 3

Problem 1

(b)

It's a pity to find that SQLite does not support non-linear recursion. While compared to Oracle SQL Developer, SQLite is much lite and easy to be executed in command line console. Besides, I find the Firefox SQLite add-on item is magic for GUI operation.

Access to a database in Python and R refers to some template statement commands. I find it amazing that a simple query or a self-defined function can be utilized to query the results. Especially, I need to remember to close the database connection when done in Python and R.

(c)

1.

SELECT name FROM Site;

```
sqlite> SELECT name FROM Site;
name
----------
DR-1
DR-3
MSK-4
```

2.

(a)

SELECT DISTINCT dated FROM Visited;

```
sqlite> SELECT DISTINCT dated FROM Visited;
dated
----------
1927-02-08
1927-02-10
1930-01-07
1930-01-12
1930-02-26

1932-01-14
1932-03-22
```

(b)

SELECT personal || " " || family AS full_name FROM Person

ORDER BY family;

```
sqlite> SELECT personal || " " || family AS full_name FROM Person
   ...> ORDER BY family;
full_name
--------------
Frank Danforth
William Dyer
Anderson Lake
Frank Pabodie
Valentina Roer
```

3.

SELECT * FROM Survey

WHERE (reading < 0 OR reading > 1) AND quant='sal';

```
sqlite> SELECT * FROM Survey
   ...> WHERE (reading < 0 OR reading > 1) AND quant='sal';
taken       person      quant       reading
----------  ----------  ----------  ----------
752         roe         sal         41.6
837         roe         sal         22.5
```

4.

(a)

SELECT reading/100 FROM Survey

WHERE person = 'roe' AND quant='sal';

```
sqlite> SELECT reading/100 FROM Survey
   ...> WHERE person='roe' AND quant='sal';
reading/100
-----------
0.416
0.225
```

(b)

SELECT taken, person, quant, reading/100 AS reading FROM Survey

WHERE person = 'roe' AND quant='sal'

UNION

SELECT * FROM Survey

WHERE (person != 'roe' OR person is null) AND quant='sal';

```
sqlite> SELECT taken, person, quant, reading/100 AS reading FROM Survey
   ...> WHERE person='roe' AND quant='sal'
   ...> UNION
   ...> SELECT * FROM Survey
   ...> WHERE (person!='roe' OR person is null) AND quant='sal';
taken       person      quant       reading
----------  ----------  ----------  ----------
619         dyer        sal         0.13
622         dyer        sal         0.09
734         lake        sal         0.05
735                     sal         0.06
751         lake        sal         0.1
752         lake        sal         0.09
752         roe         sal         0.416
837         lake        sal         0.21
837         roe         sal         0.225
```

(c)

SELECT DISTINCT substr(site,0,instr(site,'-')) AS major_site FROM Visited;

```
sqlite> SELECT DISTINCT substr(site,0,instr(site,'-')) AS major_site from Visited;
major_site
----------
DR
MSK
```

5.

(a)

SELECT * FROM Visited WHERE dated is not null ORDER BY dated;

```
sqlite> SELECT * FROM Visited WHERE dated is not null ORDER BY dated;
id          site        dated
----------  ----------  ----------
619         DR-1        1927-02-08
622         DR-1        1927-02-10
734         DR-3        1930-01-07
735         DR-3        1930-01-12
751         DR-3        1930-02-26
837         MSK-4       1932-01-14
844         DR-1        1932-03-22
```

(b)

To use sentinel values to mark missing data rather than null is another way to create the databases. It simplifies the format restriction for that attribute when creating tables, since the sentinel value has the same format as other normal values. In this way, the "not null" restriction could also be added to that attribute when creating tables, which could remind users to input values for that attribute. Besides, using sentinel values also simplifies the not null clause restriction when writing SQL queries. Sometimes users may forget to add the "is not null" clause, and unsatisfied results may be produced. While, users may not need to consider this issue when using sentinel values instead.

However, using sentinel values can also introduce some burdens. Especially when performing some mathematical computations, those sentinel values may also be included just as normal values, which will indicate incorrect results. In other words, those sentinel values should be excluded first when doing some

computations. If null values are used instead of sentinel values, aggregation functions will ignore null values automatically. Users need to pay special attention to the exact values they use as sentinel values, since the sentinel values may not easy to be identified even if they are abnormal.

6.

(a)

SELECT count(reading), avg(reading) FROM Survey

WHERE person = ‘pb’ AND quant = ‘temp’;

```
sqlite> SELECT count(reading), avg(reading) FROM Survey
   ...> WHERE person = 'pb' AND quant = 'temp';
count(reading)  avg(reading)
--------------  ------------
2               -20.0
```

(b)

```
sqlite> SELECT reading - avg(reading) FROM Survey WHERE quant='rad';
reading - avg(reading)
----------------------
4.6875
```

This query actually produces the difference between one individual radiation reading and the average of all the radiation readings. That one individual is the last one presented in the Survey table, as is shown below.

```
sqlite> SELECT reading, avg(reading) FROM Survey WHERE quant='rad';
reading     avg(reading)
----------  ------------
11.25       6.5625
```

It happens because the aggregation function “avg(reading)” should return one value as the result. Since this single value result appears juxtaposed with another attribute in the SELECT clause, the result of the whole query should also return only one row.

The average reading should be regarded as a constant when calculating the differences. So, an inner SELECT clause should be embedded. The correct SQL query should be:

SELECT reading-(SELECT avg(reading) FROM Survey WHERE quant=’rad’) AS difference

FROM Survey

WHERE quant=’rad’;

```
sqlite> SELECT reading-(SELECT avg(reading) FROM Survey WHERE quant='rad') AS difference FROM Survey WHERE quant='rad';
difference
----------
3.2575
1.2375
1.8475
0.6575
-2.2125
-4.3725
-5.1025
4.6875
```

(c)

SELECT group_concat(personal || ‘ ‘ || family, ‘, ‘) AS full_name

FROM (SELECT personal, family FROM Person ORDER BY family);

```
sqlite> SELECT group_concat(personal || ' ' || family,', ') AS full_name
   ...> FROM (SELECT personal, family FROM Person ORDER BY family);
full_name
--------------------------------------------------------------------------------
Frank Danforth, William Dyer, Anderson Lake, Frank Pabodie, Valentina Roerich
```

7.

(a)

SELECT Visited.site, Survey.reading

FROM Visited, Survey

WHERE Visited.id = Survey.taken

AND quant=’rad’ AND site=’DR-1’;

```
sqlite> SELECT Visited.site, Survey.reading
   ...> FROM Visited, Survey
   ...> WHERE Visited.id = Survey.taken
   ...> AND quant='rad'
   ...> AND site='DR-1';
site        reading
----------  ----------
DR-1        9.82
DR-1        7.8
DR-1        11.25
```

(b)

SELECT DISTINCT site, personal, family

FROM Visited, Person, Survey

WHERE Visited.id=Survey.taken AND Person.id=Survey.person

AND personal=’Frank’;

```
sqlite> SELECT DISTINCT site, personal, family
   ...> FROM Visited, Person, Survey
   ...> WHERE Visited.id = Survey.taken AND Person.id = Survey.person
   ...> AND personal='Frank';
site        personal    family
----------  ----------  ----------
DR-3        Frank       Pabodie
```

(c)

SELECT name, lat, long, dated, personal, family, quant, reading

FROM Person, Site, Survey, Visited

WHERE Person.id=Survey.person

AND Site.name=Visited.site

AND Survey.taken=Visited.id

AND dated is not null

ORDER BY dated;

```
sqlite> SELECT name, lat, long, dated, personal, family, quant, reading
   ...> FROM Person, Site, Survey, Visited
   ...> WHERE Person.id = Survey.person
   ...> AND Site.name = Visited.site
   ...> AND Survey.taken = Visited.id
   ...> AND dated is not null
   ...> ORDER BY dated;
name        lat         long        dated       personal    family      quant       reading
----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------
DR-1        -49.85      -128.57     1927-02-08  William     Dyer        rad         9.82
DR-1        -49.85      -128.57     1927-02-08  William     Dyer        sal         0.13
DR-1        -49.85      -128.57     1927-02-10  William     Dyer        rad         7.8
DR-1        -49.85      -128.57     1927-02-10  William     Dyer        sal         0.09
DR-3        -47.15      -126.72     1930-01-07  Anderson    Lake        sal         0.05
DR-3        -47.15      -126.72     1930-01-07  Frank       Pabodie     rad         8.41
DR-3        -47.15      -126.72     1930-01-07  Frank       Pabodie     temp        -21.5
DR-3        -47.15      -126.72     1930-01-12  Frank       Pabodie     rad         7.22
DR-3        -47.15      -126.72     1930-02-26  Anderson    Lake        sal         0.1
DR-3        -47.15      -126.72     1930-02-26  Frank       Pabodie     rad         4.35
DR-3        -47.15      -126.72     1930-02-26  Frank       Pabodie     temp        -18.5
MSK-4       -48.87      -123.4      1932-01-14  Anderson    Lake        rad         1.46
MSK-4       -48.87      -123.4      1932-01-14  Anderson    Lake        sal         0.21
MSK-4       -48.87      -123.4      1932-01-14  Valentina   Roerich     sal         22.5
DR-1        -49.85      -128.57     1932-03-22  Valentina   Roerich     rad         11.25
```

Problem 2

The table in the database is Publication(pid, authors, year, title, journal, vol, no, fp, lp, publisher).

```
C:\Documents\GSLIS\590 Data Cleaning\Assignment 3>sqlite3 publication.db
SQLite version 3.17.0 2017-02-13 16:02:40
Enter ".help" for usage hints.
sqlite> .tables
sqlite> CREATE TABLE IF NOT EXISTS Publication(
   ...> pid TEXT,
   ...> authors TEXT,
   ...> year NUMERIC,
   ...> title TEXT,
   ...> journal TEXT,
   ...> vol NUMERIC,
   ...> no NUMERIC,
   ...> fp NUMERIC,
   ...> lp NUMERIC,
   ...> publisher TEXT);
sqlite> .tables
Publication
```

```
sqlite> .mode column
sqlite> .header on
sqlite> select * from Publication
   ...> ;
sqlite> INSERT INTO Publication VALUES ('6755', 'hyatt', 1872, 'fossil', 'bullmcz', 5, 5, 91, 9, 'publisher1');
sqlite> INSERT INTO Publication VALUES ('2580', 'rolfe', 1962, 'phyllocarid', 'breviora', 151, 151, 4, 6, 'mcz');
sqlite> INSERT INTO Publication VALUES ('2044', 'bather', 1934, 'chelonechinus', 'gsa', 45, 4, 808, 832, null);
sqlite> INSERT INTO Publication VALUES ('4407', 'kummel', 1969, 'ammonoids', 'bullmcz', 137, 3, 476, null, 'publisher2');
```

```
sqlite> INSERT INTO Publication VALUES ('4407', 'doe', 2015, 'foobar', 'bullmcz', 10, 1, 10, 1, null);
sqlite> select * from Publication;
pid         authors     year        title       journal     vol         no          fp          lp          publisher
----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------
6755        hyatt       1872        fossil      bullmcz     5           5           91          9           publisher1
2580        rolfe       1962        phyllocari  breviora    151         151         4           6           mcz
2044        bather      1934        chelonechi  gsa         45          4           808         832
4407        kummel      1969        ammonoids   bullmcz     137         3           476                     publisher2
4407        doe         2015        foobar      bullmcz     10          1           10          1
```

(a)

(FD-1)

SELECT P1.* FROM Publication P1, Publication P2

WHERE P1.pid = P2.pid

AND ((P1.authors != P2.authors) OR (P1.year != P2.year) OR (P1.title != P2.title) OR (P1.journal != P2.journal) OR (P1.vol != P2.vol) OR (P1.no != P2.no) OR (P1.fp != P2.fp) OR (P1.lp != P2.lp) OR (P1.publisher != P2.publisher));

```
sqlite> SELECT P1.* FROM Publication P1, Publication P2
   ...> WHERE P1.pid = P2.pid
   ...> AND ((P1.authors != P2.authors) OR (P1.year != P2.year) OR (P1.title != P2.title) OR (P1.journal != P2.journal) OR (P1.vol != P2.
vol) OR (P1.no != P2.no) OR (P1.fp != P2.fp) OR (P1.lp != P2.lp) OR (P1.publisher != P2.publisher));
pid         authors     year        title       journal     vol         no          fp          lp          publisher
----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------  ----------
4407        kummel      1969        ammonoids   bullmcz     137         3           476                     publisher2
4407        doe         2015        foobar      bullmcz     10          1           10          1
```

(FD-2)

SELECT P1.journal, P1.publisher, P2.publisher FROM Publication P1, Publication P2

WHERE P1.journal = P2.journal AND P1.publisher != P2.publisher;

```
sqlite> SELECT P1.journal, P1.publisher, P2.publisher FROM Publication P1, Publication P2
   ...> WHERE P1.journal = P2.journal AND P1.publisher != P2.publisher;
journal     publisher   publisher
----------  ----------  ----------
bullmcz     publisher1  publisher2
bullmcz     publisher2  publisher1
```

(NC-1)

I suppose null value for a page number is not a violation.

SELECT P1.pid, P1.fp, P1.lp

FROM Publication P1, Publication P2

WHERE P1.pid = P2.pid AND P1.fp = P2.fp AND P1.lp = P2.lp

AND P1.fp > P2.lp;

```
sqlite> SELECT P1.pid, P1.fp, P1.lp
   ...> FROM Publication P1, Publication P2
   ...> WHERE P1.pid = P2.pid AND P1.fp = P2.fp AND P1.lp = P2.lp
   ...> AND P1.fp > P2.lp;
pid         fp          lp
----------  ----------  ----------
6755        91          9
4407        10          1
```

(b)

```
CREATE TABLE Cites(
pid1 TEXT,
pid2 TEXT);
```

```
sqlite> select * from Cites;
pid1        pid2
----------  ----------
4711        2020
4711        3799
3799        2580
2580        2044
2044        2580
```

(ID)

SELECT Cites.pid2 FROM Cites

WHERE NOT EXISTS (SELECT pid FROM Publication WHERE Cites.pid2 = Publication.pid);

```
sqlite> select * from cites;
pid1        pid2
----------  ----------
4711        2020
4711        3799
3799        2580
2580        2044
2044        2580
sqlite> select pid from Publication;
pid
----------
6755
2580
2044
4407
4407
sqlite> SELECT Cites.pid2 FROM Cites
   ...> WHERE NOT EXISTS (SELECT pid FROM Publication WHERE Cites.pid2 = Publication.pid);
pid2
----------
2020
3799
```

(NC-2)

SELECT pid1, pid2, P1.year, P2.year

FROM Cites, Publication P1, Publication P2

WHERE pid1=P1.pid AND pid2=P2.pid

AND P1.year<P2.year;

```
sqlite> SELECT pid1, pid2, P1.year, P2.year
   ...> FROM Cites, Publication P1, Publication P2
   ...> WHERE pid1=P1.pid AND pid2=P2.pid
   ...> AND P1.year<P2.year;
pid1        pid2        year        year
----------  ----------  ----------  ----------
2044        2580        1934        1962
```

(b)

Beginner SQL Tutorial: http://beginner-sql-tutorial.com/sql-integrity-constraints.htm

Constraints can be defined in two ways:

1) The constraints can be specified immediately after the column definition. This is called column-level definition. In SQLite or other relational database developers, some key words such as “PRIMARY KEY”, and “REFERENCES” can be added after the name of the attribute when writing the **creating tables clause**.
2) The constraints can be specified after all the columns are defined. This is called table-level definition.

- SQL Primary Key:
  - This constraint defines a column or combination of columns which uniquely identifies each row in the table.
  - Syntax to define a Primary key at column level:
    - column name datatype [CONSTRAINT constraint_name] **PRIMARY KEY**
  - Syntax to define a Primary key at table level:
    - [CONSTRAINT constraint_name] **PRIMARY KEY** (column_name1,column_name2,..)

- SQL Foreign Key or Referential Integrity:
  - Syntax to define a Foreign key at column level:
    - [CONSTRAINT constraint_name] **REFERENCES** Referenced_Table_name(column_name)
  - Syntax to define a Foreign key at table level:
    - [CONSTRAINT constraint_name] **FOREIGN KEY**(column_name) **REFERENCES** referenced_table_name(column_name)

- SQL Not Null Constraint:
  - Syntax to define a Not Null constraint:
    - [CONSTRAINT constraint name] **NOT NULL**

- SQL Unique Key:
  - This constraint ensures that a column or a group of columns in each row have a distinct value. A column(s) can have a null value but the values cannot be duplicated.
  - Syntax to define a Unique key at column level:
    - [CONSTRAINT constraint_name] **UNIQUE**
  - Syntax to define a Unique key at table level:
    - [CONSTRAINT constraint_name] **UNIQUE**(column_name)

- SQL Check Constraint:
  - This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.
  - Syntax to define a Check constraint:
    - [CONSTRAINT constraint_name] **CHECK** (condition)
    - E.g. `gender char(1) CHECK (gender in ('M','F'))`