Assignment 3

# Problem 1

(b)

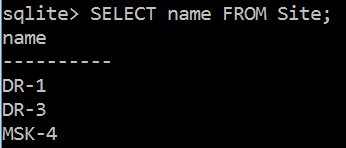
It’s a pity to find that SQLite does not support non-linear recursion. While compared to Oracle SQL Developer, SQLite is much lite and easy to be executed in command line console. Besides, I find the Firefox SQLite add-on item is magic for GUI operation.

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.

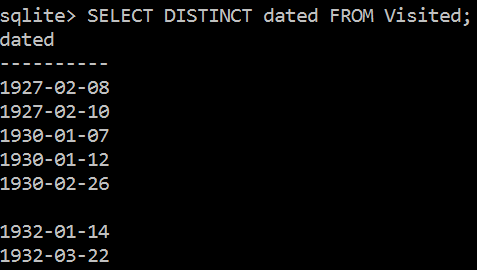
SELECT name FROM Site;



2.

(a)

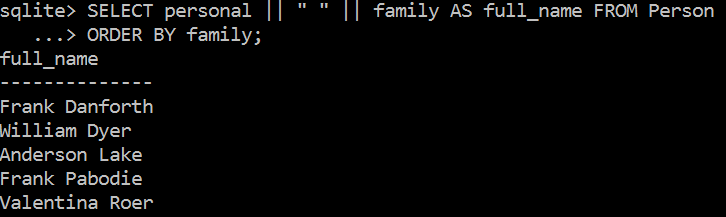
SELECT DISTINCT dated FROM Visited;



(b)

SELECT personal || “ ” || family AS full\_name FROM Person

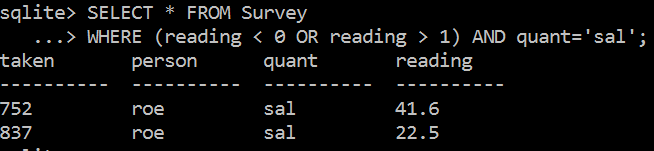
ORDER BY family;



3.

SELECT \* FROM Survey

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

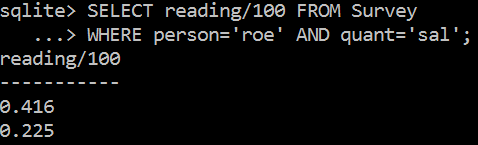


4.

(a)

SELECT reading/100 FROM Survey

WHERE person = ‘roe’ AND quant=’sal’;



(b)

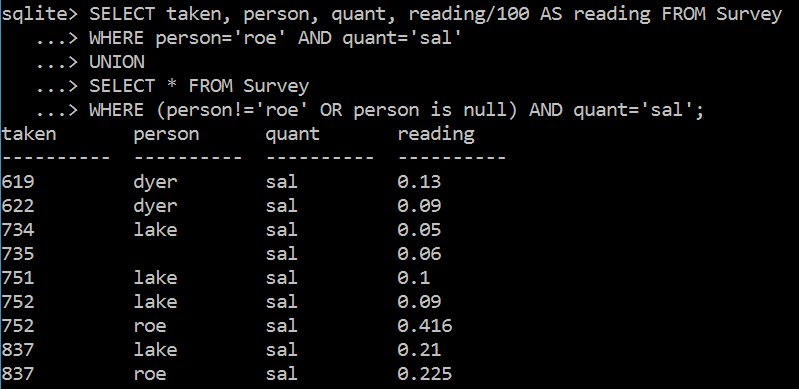
SELECT taken, person, quant, reading/100 AS reading FROM Survey

WHERE person = ‘roe’ AND quant=’sal’

UNION

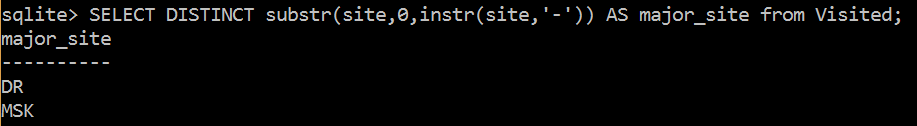
SELECT \* FROM Survey

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



(c)

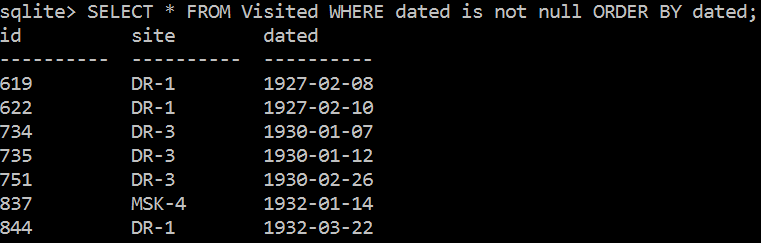
SELECT DISTINCT substr(site,0,instr(site,’-’)) AS major\_site FROM Visited;



5.

(a)

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



(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.

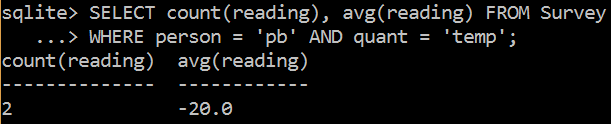
However, using sentinel values can also introduce some burdens. Especially when performing some mathematical computations, those sentinel values may also be included just as normal values, which will indicate incorrect results. In other words, those sentinel values should be excluded first when doing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

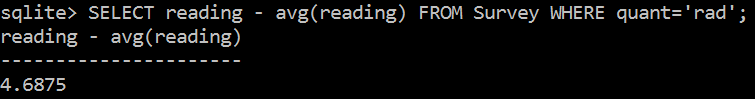
(a)

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

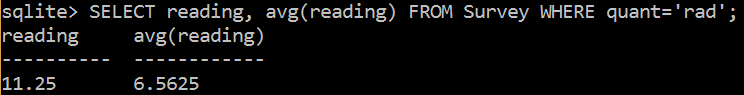
WHERE person = ‘pb’ AND quant = ‘temp’;



(b)



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.



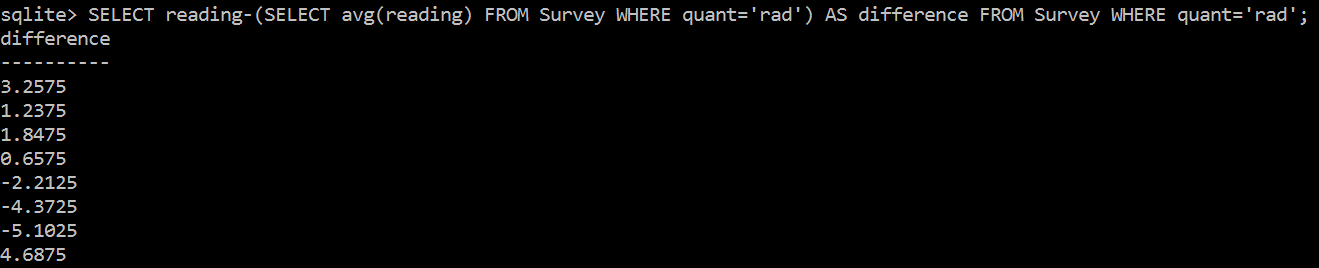
It happens because the aggregation function “avg(reading)” should return one value as the result. Since this single value result appears juxtaposed with another attribute in the SELECT clause, the result of the whole query should also return only one row.

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

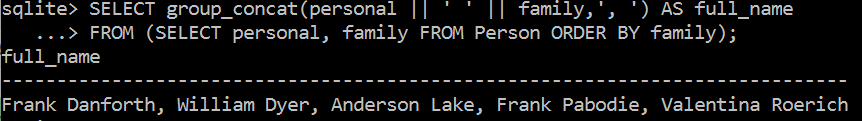
WHERE quant=’rad’;



(c)

SELECT group\_concat(personal || ‘ ‘ || family, ‘, ‘) AS full\_name

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



7.

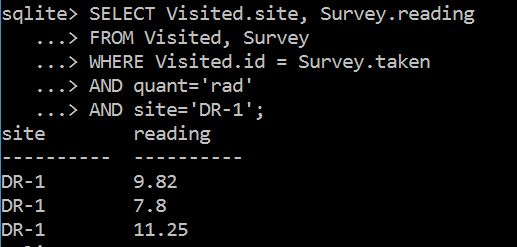
(a)

SELECT Visited.site, Survey.reading

FROM Visited, Survey

WHERE Visited.id = Survey.taken

AND quant=’rad’ AND site=’DR-1’;



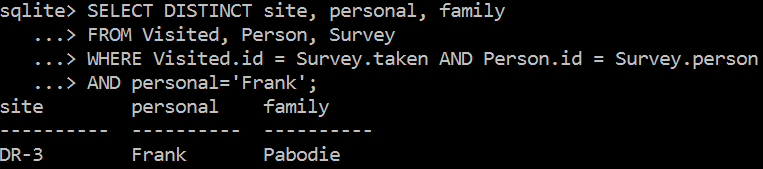
(b)

SELECT DISTINCT site, personal, family

FROM Visited, Person, Survey

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

AND personal=’Frank’;



(c)

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

FROM Person, Site, Survey, Visited

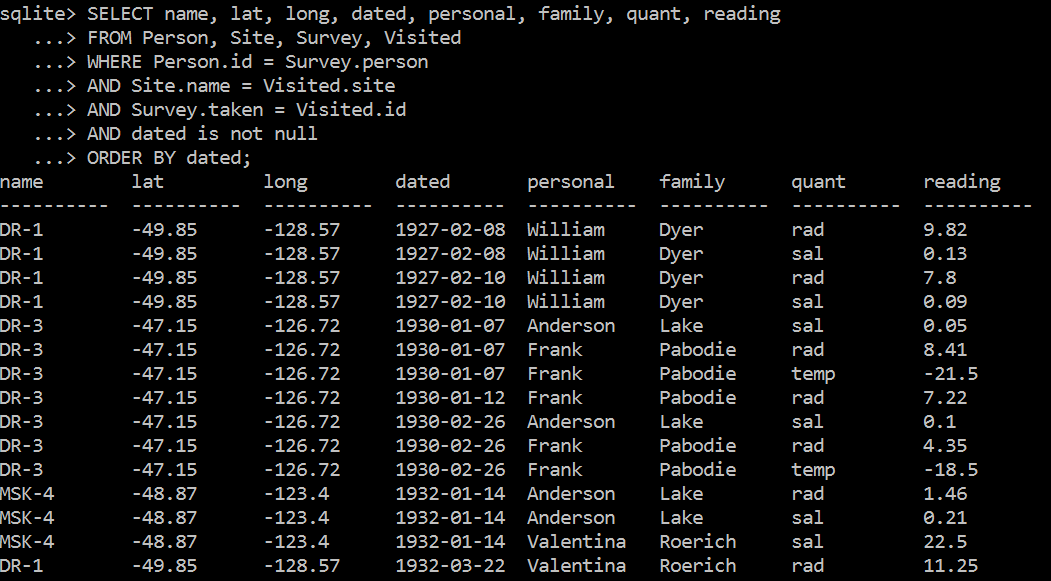
WHERE Person.id=Survey.person

AND Site.name=Visited.site

AND Survey.taken=Visited.id

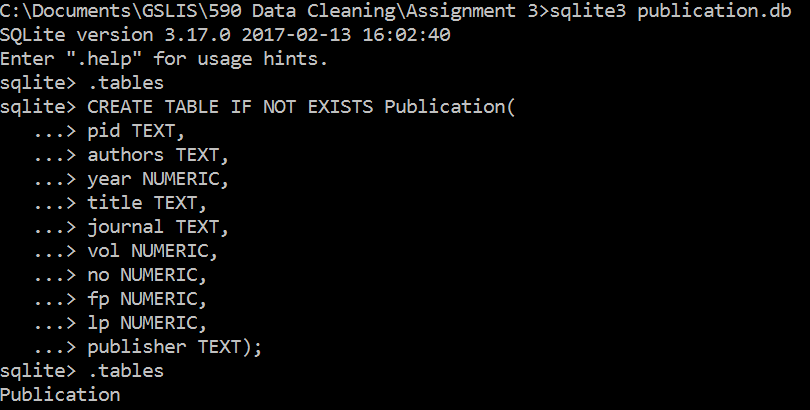
AND dated is not null

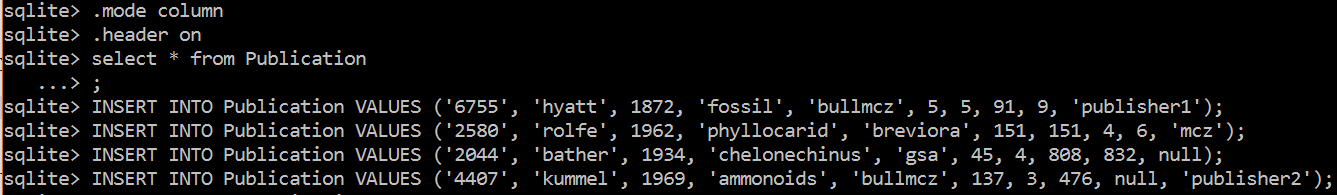
ORDER BY dated;

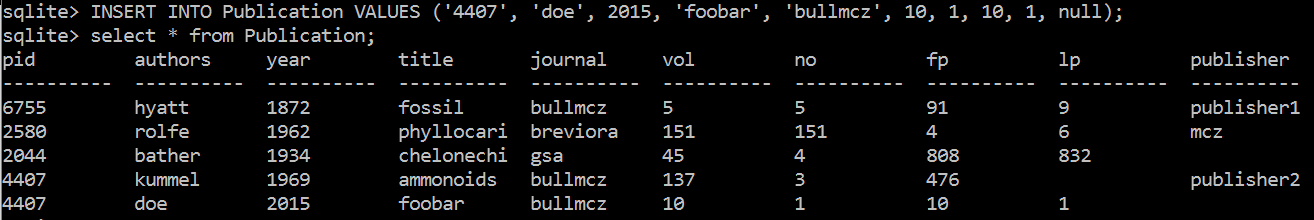


# Problem 2

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







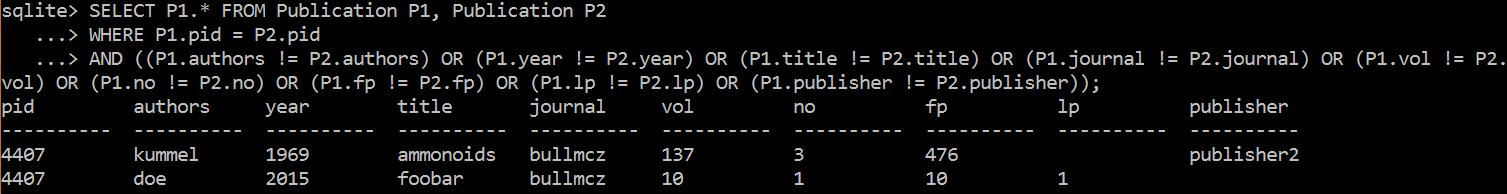
(a)

(FD-1)

SELECT P1.\* FROM Publication P1, Publication P2

WHERE P1.pid = P2.pid

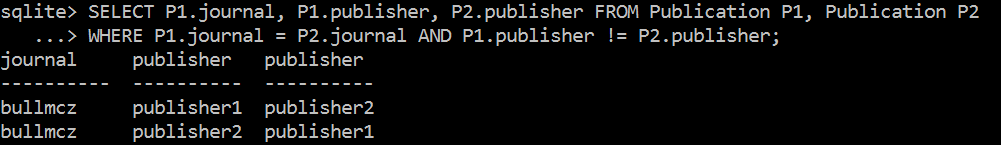
AND ((P1.authors != P2.authors) OR (P1.year != P2.year) OR (P1.title != P2.title) OR (P1.journal != P2.journal) OR (P1.vol != P2.vol) OR (P1.no != P2.no) OR (P1.fp != P2.fp) OR (P1.lp != P2.lp) OR (P1.publisher != P2.publisher));



(FD-2)

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

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



(NC-1)

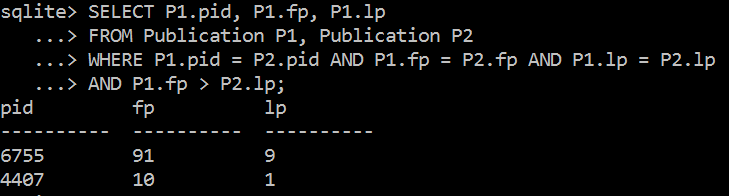
I suppose null value for a page number is not a violation.

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

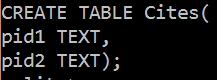
FROM Publication P1, Publication P2

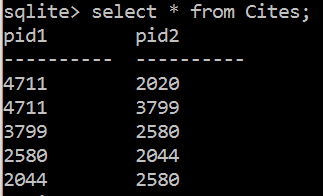
WHERE P1.pid = P2.pid AND P1.fp = P2.fp AND P1.lp = P2.lp

AND P1.fp > P2.lp;



(b)

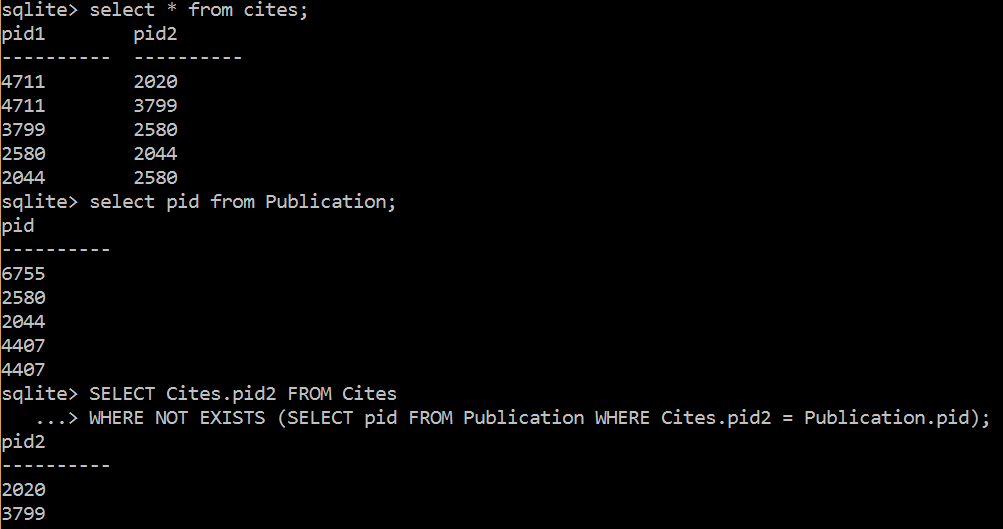




(ID)

SELECT Cites.pid2 FROM Cites

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



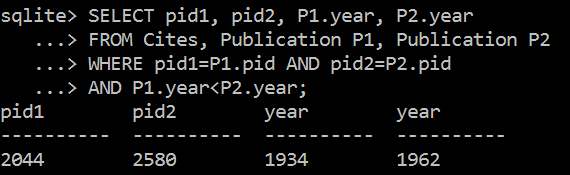
(NC-2)

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;



(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'))