```
import sqlite3
import pandas as pd
connection = sqlite3.connect('family.db')
cursor = connection.cursor()
```

First create a table called parents. It has two columns: 'parent' and 'child'. The first column indicates the parent of the child in the second column. We will use a new form of CREATE TABLE expression to produce this table.

```
cmd = """CREATE TABLE parents (
parent VARCHAR(20),
child VARCHAR(20));
"""
cursor.execute(cmd)
```

```
<sqlite3.Cursor at 0x7d63b08dbe40>
```

```
cmd = """INSERT INTO parents (parent, child)
VALUES ("abraham", "barack") UNION
VALUES ("abraham", "clinton") UNION
VALUES ("delano", "herbert") UNION
VALUES ("fillmore", "abraham") UNION
VALUES ("fillmore", "delano") UNION
VALUES ("fillmore", "grover") UNION
VALUES ("eisenhower", "fillmore");"""
cursor.execute(cmd)
```

```
<sqlite3.Cursor at 0x7d63b08dbe40>
```

Q1 Simple SELECTS (on the parents table)

1. SELECT all records in the table

```
cmd = """SELECT * FROM parents;"""
pd.read_sql_query(cmd, con = connection)
```

	parent	child
0	abraham	barack
1	abraham	clinton
2	delano	herbert
3	eisenhower	fillmore
4	fillmore	abraham
5	fillmore	delano
6	fillmore	grover

2. SELECT child and parent, where abraham is the parent

```
cmd = """SELECT child,parent FROM parents
WHERE parent='abraham';"""
pd.read_sql_query(cmd, con = connection)
```

	child	parent
0	barack	abraham
1	clinton	abraham

3. SELECT all children that have an 'e' in their name (hint: use LIKE and '%e%').

```
cmd = """SELECT child FROM parents
WHERE child LIKE '%e%'"""
pd.read_sql_query(cmd, con = connection)
```

	child
0	herbert
1	fillmore
2	delano
3	grover

4. SELECT all unique parents (use SELECT DISTINCT) and order them by name, descending order (i.e. fillmore first)

```
cmd = """SELECT DISTINCT parent
FROM parents
ORDER BY parent DESC"""
pd.read_sql_query(cmd, con = connection)
```

	parent
0	fillmore
1	eisenhower
2	delano
3	abraham

5. SELECT all dogs that are siblings (one-to-one relations). Only show a sibling pair once. To do this you need to select two times from the parents table.

```
cmd = """SELECT p1.child as child1, p2.child as child2
FROM parents p1 JOIN parents p2
ON (p1.parent = p2.parent) AND (p1.child > p2.child)"""
pd.read_sql_query(cmd, con = connection)
```

	child1	child2
0	clinton	barack
1	delano	abraham
2	grover	abraham
3	grover	delano

Q2 Joins

Create a new table called dogs, which indicates the fur type of every dog. In the image above:

- long haired dogs = red dashed box
- curly haired dogs = black fluffy box
- short haired dogs = grey dotted box

```
cmd = """CREATE TABLE dogs AS
SELECT "abraham" AS name, "long" AS fur UNION
SELECT "barack", "short" UNION
SELECT "clinton", "long" UNION
SELECT "delano", "long" UNION
SELECT "eisenhower", "short" UNION
SELECT "fillmore", "curly" UNION
SELECT "grover", "short" UNION
SELECT "herbert", "curly";
"""
cursor.execute(cmd)
```

```
<sqlite3.Cursor at 0x7d63b08dbe40>
```

1. COUNT the number of short haired dogs

```
cmd = """SELECT COUNT(*) as NoOfShortHairedDogs
FROM dogs
WHERE fur = 'short';"""
pd.read_sql_query(cmd, con = connection)
```

	NoOfShortHairedDogs
0	3

2. JOIN tables parents and dogs and SELECT the parents of curly dogs.

```
cmd = """SELECT p.parent
FROM parents p JOIN dogs d
WHERE p.child = d.name and d.fur='curly';"""
pd.read_sql_query(cmd, con = connection)
```

	parent
0	eisenhower
1	delano

3. JOIN tables parents and dogs, and SELECT the parents and children that have the same fur type. Only show them once.

```
cmd = """SELECT p.parent, p.child
FROM parents p JOIN dogs d1
ON p.child = d1.name
JOIN dogs d2
ON p.parent = d2.name
WHERE d1.fur = d2.fur;"""
pd.read_sql_query(cmd, con = connection)
```

	parent	child
0	abraham	clinton

Q3 Aggregate functions, numerical logic and grouping

Create a new table with many different animals. The table includes the animal's kind, number of legs and weight

```
cmd = """CREATE table animals AS
SELECT "dog" AS kind, 4 AS legs, 20 AS weight UNION
SELECT "cat" , 4 , 10 UNION
SELECT "ferret" , 4 , 10 UNION
SELECT "parrot" , 2 , 6 UNION
SELECT "penguin" , 2 , 10 UNION
SELECT "t-rex" , 2 , 12000;"""
cursor.execute(cmd)
```

```
<sqlite3.Cursor at 0x7d63b08dbe40>
```

1. SELECT the animal with the minimum weight. Display kind and min_weight

```
cmd = """SELECT kind, weight
FROM animals
WHERE weight = (SELECT MIN(weight) FROM animals);"""
pd.read_sql_query(cmd, con = connection)
```

```
kind  weight
cmd = """SELECT kind, weight
FROM animals
ORDER BY weight
LIMIT 1;"""
pd.read_sql_query(cmd, con = connection)
```

	kind	weight
0	parrot	6

2. Use the aggregate function AVG to display a table with the average number of legs and the average weight

```
cmd = """SELECT AVG(legs) as avg_legs, AVG(weight) as avg_weight
FROM animals;"""
pd.read_sql_query(cmd, con = connection)
```

	avg_legs	avg_weight
0	3.0	2009.333333

3. SELECT the animal kind(s) that have more than two legs, but weighs less than 20. Display kind, weight, legs.

```
cmd = """SELECT kind,weight,legs
FROM animals
WHERE legs > 2 AND weight < 20;"""
pd.read_sql_query(cmd, con = connection)
```

	kind	weight	legs
0	cat	10	4
1	ferret	10	4

4. SELECT the average weight for all the animals with 2 legs and the animals with 4 legs (by using GROUP BY)

```
cmd = """SELECT legs, AVG(weight) as avg_weight
FROM animals
GROUP BY legs
HAVING legs in (2,4);"""
pd.read_sql_query(cmd, con = connection)
```

	legs	avg_weight
0	2	4005.333333
1	4	13.333333