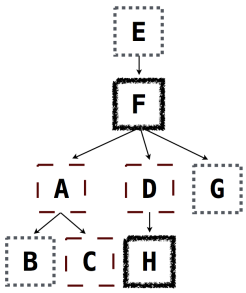
Exercises: Let's play with Dogs (& SQL)

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.

```
CREATE TABLE parents AS
  SELECT "abraham" AS parent, "barack" AS child UNION
  SELECT "abraham", "clinton" UNION
  SELECT "delano", "herbert" UNION
  SELECT "fillmore", "abraham" UNION
  SELECT "fillmore", "delano" UNION
  SELECT "fillmore", "grover" UNION
  SELECT "eisenhower", "fillmore";
```

Picture of the Dog Family Tree (illustration of parents table)

(A = abrham, B = barack, etc.)



Q1 Simple SELECTS (on the parents table)

1. SELECT all records in the table.
2. SELECT child and parent, where abraham is the parent.
3. SELECT all children that have an 'e' in their name (hint: use LIKE and '%e%').
4. SELECT all unique parents (use SELECT DISTINCT) and order them by name, descending order (i.e. fillmore first)
5. **Difficult***: 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.

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, and short haired dogs = grey dotted box.

Create the table by running:

```
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";
```

1. COUNT the number of short haired dogs
2. JOIN tables parents and dogs and SELECT the parents of curly dogs.
3. **Difficult**: JOIN tables parents and dogs, and SELECT the parents and children that have the same fur type. Only show them once.

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. Create it by running:

```
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;
```

1. SELECT the animal with the minimum weight. Display kind and min_weight.
2. Use aggregate function AVG to display a table with the average number of legs and the average weight.
3. SELECT the animal kind(s) that have more than two legs, but weighs less than 20. Display kind, weight, legs.
4. SELECT the average weight for all the animals with 2 legs and the animals with 4 legs (by using GROUP BY).

.

These exercises are inspired by the Lectures in CS61A (Fall 2014).

Your Solution Here

```
# Import required packages

import sqlite3
import pandas as pd
```

```
# list files in working directory
!ls
```

```
     sample_data
```

```
# Open the connnection to a database file and create it if it doesn't exist
connection = sqlite3.connect('dogdata.db')
```

```
!ls
```

```
     dogdata.db  sample_data
```

```
# Create a cursor object to traverse the database
cursor = connection.cursor()
```

Q1 - Dog Family Tree

```
# Create a table called parents

sql_command = '''
CREATE TABLE parents AS
  SELECT "abraham" AS parent, "barack" AS child UNION
  SELECT "abraham", "clinton" UNION
  SELECT "delano", "herbert" UNION
  SELECT "fillmore", "abraham" UNION
  SELECT "fillmore", "delano" UNION
  SELECT "fillmore", "grover" UNION
  SELECT "eisenhower", "fillmore";'''

cursor.execute(sql_command)
```

```
     <sqlite3.Cursor at 0x7a9c625a0a40>
```

```
# Commit the query

connection.commit()
```

1. SELECT all records in the table.

```
# Retrieve using SELECT statement

pd.read_sql_query('SELECT * FROM parents', 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.

```
# Retrieve using SELECT statement and filter using WHERE clause

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

	child	parent
0	barack	abraham
1	clinton	abraham
2	abraham	fillmore

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

```
# Retrieve using SELECT statement and LIKE operator

pd.read_sql_query('SELECT child FROM parents WHERE child LIKE "%e%";', 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)

```
# Retrieve using SELECT statement and sort using ORDER BY keyword

pd.read_sql_query('SELECT DISTINCT parent FROM parents ORDER BY parent DESC', 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.

```
# Perform cross join between the parents table and itself by listing both tables in the FROM clause

pd.read_sql_query('''SELECT DISTINCT p1.child AS sibling_1, p2.child AS sibling_2
                        FROM parents p1, parents p2
                        WHERE p1.parent = p2.parent
                        AND p1.child < p2.child''', con = connection)
```

	sibling_1	sibling_2
0	barack	clinton
1	abraham	delano

Q2 Joins

```
# Create a new table called dogs which indicates the fur type of every dog

sql_command = '''
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(sql_command)
```

```
<sqlite3.Cursor at 0x7a9c625a0a40>
```

```
# Commit the query

connection.commit()
```

```
pd.read_sql_query('SELECT * FROM dogs', con = connection)
```

	name	fur
0	abraham	long
1	barack	short
2	clinton	long
3	delano	long
4	eisenhower	short
5	fillmore	curly
6	grover	short
7	herbert	curly

1. COUNT the number of short haired dogs

```
# Use COUNT() function

pd.read_sql_query('SELECT fur AS fur_type, Count(*) AS num_dogs FROM dogs WHERE fur = "short"', con = connection)
```

	fur_type	num_dogs
0	short	3

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

```
# Peform Inner Join on parents and dogs

pd.read_sql_query('SELECT p.parent FROM parents p JOIN dogs d ON p.child = d.name WHERE fur = "curly"', 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.

```

```

```
# Peform Inner Join on parents and dogs and also use Subquery

pd.read_sql_query('''SELECT DISTINCT p.parent, p.child
                     FROM parents p
                     JOIN dogs d ON p.child = d.name
                     WHERE d.fur = (SELECT fur FROM dogs WHERE name = p.parent)''', con = connection)
```

	parent	child
0	abraham	clinton

Q3 - Aggregate functions, numerical logic and grouping

```
# Create a new table animals with the animal's kind, number of legs and weight.

sql_command = '''
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(sql_command)
```

```
<sqlite3.Cursor at 0x7a9c625a0a40>
```

```
# Commit the query

connection.commit()
```

```
pd.read_sql_query('SELECT * FROM animals', con = connection)
```

	kind	legs	weight
0	cat	4	10
1	dog	4	20
2	ferret	4	10
3	parrot	2	6
4	penguin	2	10
5	t-rex	2	12000

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

```
# Use MIN() function

pd.read_sql_query('SELECT kind, MIN(weight) AS min_weight FROM animals', con = connection)
```

	kind	min_weight
0	parrot	6

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

```
# Use AVG() function

pd.read_sql_query('SELECT AVG(legs) AS avg_num_legs, AVG(weight) AS avg_weight FROM animals', con = connection)
```

	avg_num_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.

```

```

```
# Use GROUP BY statement and HAVING clause

pd.read_sql_query('SELECT kind, weight, legs FROM animals GROUP BY kind HAVING legs > 2 AND weight < 20', con = connection)
```

	kind	weight	legs
0	cat	10	4
1	ferret	10	4

OR

```
# Use WHERE clause

pd.read_sql_query('SELECT kind, weight, legs FROM animals WHERE legs > 2 AND weight < 20', 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).

```
# Use AVG() function and GROUP BY statement

pd.read_sql_query('SELECT legs, AVG(weight) AS avg_weight FROM animals GROUP BY legs', con = connection)
```

	legs	avg_weight
0	2	4005.333333
1	4	13.333333