DSC540 Week 9&10 Exercises

Name: Madhuri Basava

Date: 08/07/2023

1) Connect to the supplied petsDB Database and whether the connection is successful

Import the necessary package

```
In [1]: import sqlite3
```

```
In [2]: conn = sqlite3.connect("petsdb")  #Connect to the database
```

Define a function isConnected() to check if the connection is successful

```
In [3]: def isConnected(conn):
            try:
                conn.execute("SELECT * FROM 'persons' LIMIT 1")
                return True
            except sqlite3.ProgrammingError as e:
                print("Connected closed {}".format(e))
                return False
```

```
In [4]: print(isConnected(conn))    #Yes,the connection is successful
```

```
True
```

```
In [5]: conn.close()   #Connection is closed
```

```
In [6]: print(isConnected(conn))
```

```
Connected closed Cannot operate on a closed database.
False
```

2) Find the different age groups in the persons database

```
In [7]: conn = sqlite3.connect("petsdb")  #Connect to the databse again as it was closed be
```

```
In [8]: c = conn.cursor()
```

Display all the records in person table

In [9]:
```
persons_data = c.execute("SELECT * FROM persons")
names = [description[0] for description in persons_data.description]
print(names)  #print the headers of person table

# print all the rows from `persons` table
for row in persons_data:
    print(row)
```

```
['id', 'first_name', 'last_name', 'age', 'city', 'zip_code']
(1, 'Erica', None, 22, 'south port', 2345678)
(2, 'Jordi', None, 73, 'east port', 123456)
(3, 'Chasity', None, 70, 'new port', 76856785)
(4, 'Gregg', None, 31, 'new port', 76856785)
(5, 'Tony', 'Lindgren', 7, 'west port', 2345678)
(6, 'Cary', None, 73, 'new port', 76856785)
(7, 'Gabe', 'Thompson', 54, 'new port', 9756543)
(8, 'Francisca', None, 14, 'west port', 123456)
(9, 'Katelyn', 'Torphy', 49, 'east port', 9756543)
(10, 'Raleigh', None, 68, 'new port', 2345678)
(11, 'Maria', None, 42, 'west port', 123456)
(12, 'Mariane', None, 62, 'south port', 9756543)
(13, 'Mona', None, 44, 'south port', 76856785)
(14, 'Kayla', None, 36, 'south port', 2345678)
(15, 'Karlie', None, 35, 'west port', 123456)
(16, 'Morris', None, 71, 'west port', 76856785)
(17, 'Sandy', None, 23, 'east port', 2345678)
(18, 'Hector', None, 63, 'east port', 9756543)
(19, 'Hiram', None, 52, 'west port', 2345678)
(20, 'Tressa', None, 59, 'new port', 123456)
(21, 'Berry', None, 22, 'south port', 2345678)
(22, 'Pearline', None, 73, 'new port', 9756543)
(23, 'Maynard', None, 25, 'east port', 123456)
(24, 'Dorian', None, 40, 'east port', 123456)
(25, 'Mylene', None, 5, 'east port', 76856785)
(26, 'Lafayette', None, 34, 'new port', 2345678)
(27, 'Friedrich', 'Wunsch', 48, 'east port', 9756543)
(28, 'Domenica', 'Lakin', 13, 'west port', 76856785)
(29, 'Tara', None, 39, 'west port', 123456)
(30, 'Destiny', None, 18, 'south port', 2345678)
(31, 'Lesly', None, 31, 'west port', 123456)
(32, 'Perry', None, 19, 'south port', 76856785)
(33, 'Keyon', 'Upton', 51, 'south port', 123456)
(34, 'Jacquelyn', 'Bradtke', 54, 'south port', 123456)
(35, 'Maritza', None, 73, 'east port', 9756543)
(36, 'Ivory', 'Sawayn', 8, 'south port', 76856785)
(37, 'Grant', None, 61, 'east port', 76856785)
(38, 'Salma', 'Terry', 71, 'west port', 123456)
(39, 'Laury', None, 17, 'east port', 9756543)
(40, 'Name', None, 52, 'east port', 9756543)
(41, 'Estefania', None, 32, 'new port', 76856785)
(42, 'Destiney', None, 65, 'west port', 2345678)
(43, 'Jaquelin', None, 73, 'west port', 9756543)
(44, 'Brenda', 'Murray', 62, 'south port', 2345678)
(45, 'Alfonzo', None, 16, 'east port', 2345678)
(46, 'Lisandro', None, 11, 'new port', 76856785)
(47, 'Lurline', 'Becker', 24, 'west port', 76856785)
(48, 'Timmothy', 'Tillman', 12, 'new port', 76856785)
(49, 'Priscilla', None, 65, 'east port', 76856785)
(50, 'Elenora', None, 11, 'new port', 76856785)
(51, 'Wendell', 'Sauer', 39, 'south port', 76856785)
(52, 'Rudolph', None, 14, 'east port', 76856785)
(53, 'Gabriella', 'Kertzmann', 30, 'south port', 76856785)
(54, 'Citlalli', 'Pollich', 18, 'west port', 2345678)
(55, 'Dax', 'Romaguera', 23, 'west port', 123456)
```

```
(56, 'Ona', None, 35, 'east port', 9756543)
(57, 'Rebeca', None, 50, 'new port', 76856785)
(58, 'Lou', 'Bradtke', 71, 'south port', 2345678)
(59, 'Sigurd', None, 12, 'west port', 76856785)
(60, 'Nicklaus', 'Purdy', 72, 'new port', 2345678)
(61, 'Destiney', 'Sauer', 24, 'west port', 2345678)
(62, 'Cassidy', 'Crona', 18, 'east port', 123456)
(63, 'Alice', None, 8, 'west port', 76856785)
(64, 'Dane', None, 24, 'west port', 9756543)
(65, 'Judge', None, 17, 'south port', 76856785)
(66, 'Allene', None, 9, 'new port', 9756543)
(67, 'Jalen', None, 33, 'new port', 2345678)
(68, 'Aniyah', 'Grant', 27, 'new port', 9756543)
(69, 'Ashton', 'Schaden', 25, 'west port', 2345678)
(70, 'Myron', None, 36, 'new port', 9756543)
(71, 'Presley', 'Gleason', 58, 'new port', 9756543)
(72, 'Ephraim', 'Kihn', 67, 'south port', 123456)
(73, 'Travon', None, 16, 'south port', 2345678)
(74, 'Shayna', None, 60, 'new port', 2345678)
(75, 'Myah', None, 14, 'east port', 2345678)
(76, 'Sammy', 'Franecki', 53, 'south port', 76856785)
(77, 'Ebba', 'Kihn', 34, 'west port', 2345678)
(78, 'Leilani', 'Gislason', 53, 'south port', 123456)
(79, 'Isabel', 'Beier', 74, 'west port', 2345678)
(80, 'Jo', 'DuBuque', 68, 'west port', 76856785)
(81, 'Angel', 'Mraz', 37, 'west port', 123456)
(82, 'Letha', None, 44, 'new port', 9756543)
(83, 'Lily', 'Hegmann', 12, 'east port', 123456)
(84, 'Felton', None, 74, 'east port', 2345678)
(85, 'London', None, 66, 'east port', 9756543)
(86, 'Koby', None, 31, 'west port', 9756543)
(87, 'Golden', None, 35, 'east port', 76856785)
(88, 'Elfrieda', 'Nikolaus', 36, 'west port', 2345678)
(89, 'Anissa', None, 8, 'south port', 76856785)
(90, 'Mathew', 'Luettgen', 71, 'east port', 2345678)
(91, 'Sid', None, 22, 'west port', 123456)
(92, 'Micheal', 'Schuppe', 6, 'west port', 9756543)
(93, 'Joanie', 'Cassin', 68, 'new port', 76856785)
(94, 'Jerad', 'Pfeffer', 74, 'west port', 123456)
(95, 'Benjamin', 'Lowe', 51, 'south port', 9756543)
(96, 'Ernesto', None, 69, 'east port', 9756543)
(97, 'Josianne', None, 14, 'west port', 76856785)
(98, 'Hilma', 'Waelchi', 48, 'east port', 9756543)
(99, 'Otilia', 'Gleason', 66, 'new port', 9756543)
(100, 'Adrian', 'Schinner', 5, 'south port', 76856785)
```

Display the different age groups in the persons database (query the databse group by age)

In [10]:
```
for ppl, age in c.execute("SELECT count(*), age FROM persons GROUP BY age"):
    print("We have {} people aged {}".format(ppl, age))
```

```
We have 2 people aged 5
We have 1 people aged 6
We have 1 people aged 7
We have 3 people aged 8
We have 1 people aged 9
We have 2 people aged 11
We have 3 people aged 12
We have 1 people aged 13
We have 4 people aged 14
We have 2 people aged 16
We have 2 people aged 17
We have 3 people aged 18
We have 1 people aged 19
We have 3 people aged 22
We have 2 people aged 23
We have 3 people aged 24
We have 2 people aged 25
We have 1 people aged 27
We have 1 people aged 30
We have 3 people aged 31
We have 1 people aged 32
We have 1 people aged 33
We have 2 people aged 34
We have 3 people aged 35
We have 3 people aged 36
We have 1 people aged 37
We have 2 people aged 39
We have 1 people aged 40
We have 1 people aged 42
We have 2 people aged 44
We have 2 people aged 48
We have 1 people aged 49
We have 1 people aged 50
We have 2 people aged 51
We have 2 people aged 52
We have 2 people aged 53
We have 2 people aged 54
We have 1 people aged 58
We have 1 people aged 59
We have 1 people aged 60
We have 1 people aged 61
We have 2 people aged 62
We have 1 people aged 63
We have 2 people aged 65
We have 2 people aged 66
We have 1 people aged 67
We have 3 people aged 68
We have 1 people aged 69
We have 1 people aged 70
We have 4 people aged 71
We have 1 people aged 72
We have 5 people aged 73
We have 3 people aged 74
```

3) Which age group has maximum number of people?

Query the dabebase with group by age and order by count in the descending order to get the highest number on the top

In [11]:
```
for ppl, age in c.execute("SELECT count(*), age FROM persons GROUP BY age ORDER BY
    print("Maximum number of people {} came from {} age group".format(ppl, age))
    break
```

```
Maximum number of people 5 came from 73 age group
```

4) How many people do not have a full name (Last name is blank/null)?

In [12]:
```
res = c.execute("SELECT count(*) FROM persons WHERE last_name IS null")
for row in res:
    print(row)
```

```
(60,)
```

5) How many people has more than one pet?

Display all the records in pets table

In [13]:
```
pets_data = c.execute("SELECT * FROM pets")
names = [description[0] for description in pets_data.description]
print(names)   #print the headers of pets table

# print all the rows from `pets` table
for row in pets_data:
    print(row)
```

```
['owner_id', 'pet_name', 'pet_type', 'treatment_done']
(57, 'mani', 1.0, 0)
(80, 'tamari', None, 0)
(25, 'raba', None, 0)
(27, 'olga', None, 0)
(60, 'raba', None, 0)
(37, 'dara', 1.0, 0)
(33, 'chegal', 1.0, 0)
(16, 'dara', None, 0)
(100, 'chegal', None, 0)
(46, 'raba', None, 1)
(21, 'tamari', 1.0, 0)
(94, 'snoopy', 1.0, 1)
(100, 'bulga', None, 1)
(5, 'fenga', None, 0)
(24, 'palu', None, 0)
(16, 'fenga', 1.0, 1)
(48, 'snoopy', None, 0)
(27, 'gimir', 1.0, 0)
(67, 'raba', None, 0)
(31, 'chegal', 1.0, 1)
(14, 'unsa', 1.0, 0)
(40, 'sapi', None, 0)
(25, 'bhukbhuk', None, 0)
(42, 'dara', None, 0)
(45, 'sami', None, 1)
(27, 'bulga', 1.0, 0)
(20, 'olga', None, 0)
(8, 'chegal', None, 0)
(45, 'nani', 1.0, 0)
(56, 'deru', None, 0)
(64, 'milu', 1.0, 0)
(2, 'mani', 1.0, 0)
(36, 'dara', None, 0)
(89, 'sapi', None, 0)
(28, 'mani', 1.0, 0)
(95, 'deru', None, 0)
(26, 'mani', None, 1)
(60, 'helu', 1.0, 0)
(10, 'snoopy', 1.0, 0)
(25, 'milu', None, 0)
(6, 'milu', 1.0, 0)
(31, 'fenga', None, 0)
(18, 'deru', None, 0)
(6, 'deru', 1.0, 1)
(35, 'milu', None, 0)
(42, 'deru', None, 1)
(21, 'chegal', None, 1)
(83, 'olga', None, 0)
(69, 'raba', 1.0, 1)
(88, 'chegal', 1.0, 0)
(13, 'gimir', None, 0)
(82, 'unsa', 1.0, 1)
(27, 'unsa', None, 0)
(38, 'raba', None, 0)
(23, 'snoopy', 1.0, 1)
```

```
(71, 'bulga', None, 0)
(64, 'palu', 1.0, 0)
(43, 'tamari', None, 0)
(79, 'gimir', None, 0)
(9, 'gimir', None, 0)
(98, 'bulga', 1.0, 1)
(43, 'helu', None, 0)
(10, 'nani', 1.0, 0)
(33, 'tamari', None, 0)
(32, 'gimir', 1.0, 0)
(37, 'helu', None, 0)
(28, 'fenga', None, 0)
(57, 'mani', 1.0, 0)
(97, 'palu', 1.0, 1)
(38, 'sami', 1.0, 0)
(81, 'milu', 1.0, 0)
(2, 'gimir', 1.0, 0)
(68, 'gimir', None, 1)
(99, 'milu', None, 1)
(7, 'olga', 1.0, 0)
(3, 'tamari', None, 1)
(40, 'palu', 1.0, 0)
(27, 'bulga', 1.0, 0)
(85, 'milu', None, 0)
(9, 'palu', 1.0, 0)
(15, 'nani', 1.0, 0)
(98, 'sami', None, 0)
(90, 'mani', 1.0, 1)
(41, 'mani', 1.0, 1)
(35, 'dara', 1.0, 1)
(13, 'dara', 1.0, 0)
(12, 'unsa', None, 0)
(99, 'helu', None, 0)
(56, 'unsa', None, 1)
(91, 'unsa', None, 0)
(7, 'unsa', None, 0)
(53, 'sami', 1.0, 1)
(33, 'helu', None, 0)
(17, 'palu', 1.0, 0)
(86, 'milu', 1.0, 0)
(64, 'chegal', None, 1)
(53, 'chegal', None, 0)
(14, 'chegal', 1.0, 0)
(20, 'fenga', 1.0, 0)
(49, 'sami', 1.0, 0)
(51, 'bulga', None, 0)
(1, 'raba', None, 1)
(51, 'bhukbhuk', 1.0, 0)
(18, 'milu', None, 0)
(99, 'snoopy', 1.0, 0)
(57, 'snoopy', None, 1)
(21, 'mani', 1.0, 0)
(39, 'sapi', None, 0)
(84, 'chegal', None, 1)
(5, 'dara', 1.0, 0)
(47, 'sapi', None, 0)
```

```
(98, 'helu', 1.0, 0)
(2, 'tamari', None, 1)
(37, 'nani', None, 0)
(39, 'dara', None, 0)
(37, 'sami', None, 0)
(50, 'bulga', 1.0, 0)
(50, 'milu', None, 0)
(35, 'chegal', None, 0)
(5, 'dara', None, 0)
(40, 'chegal', None, 0)
(42, 'snoopy', None, 0)
(58, 'sapi', 1.0, 0)
(73, 'olga', None, 1)
(58, 'sapi', 1.0, 0)
(43, 'palu', None, 0)
(36, 'snoopy', 1.0, 0)
(11, 'fenga', None, 0)
(56, 'milu', None, 1)
(56, 'sapi', None, 1)
(60, 'bhukbhuk', 1.0, 0)
(7, 'helu', None, 1)
(56, 'olga', None, 0)
(10, 'sapi', 1.0, 1)
(24, 'fenga', None, 1)
(55, 'tamari', None, 0)
(67, 'palu', None, 0)
(44, 'sami', 1.0, 0)
(93, 'palu', 1.0, 0)
(50, 'tamari', None, 0)
(49, 'fenga', 1.0, 0)
(40, 'milu', 1.0, 0)
(48, 'nani', 1.0, 0)
(51, 'bhukbhuk', None, 0)
(77, 'sami', 1.0, 1)
(86, 'fenga', 1.0, 0)
(87, 'palu', 1.0, 0)
(26, 'snoopy', 1.0, 0)
(36, 'nani', 1.0, 0)
(22, 'helu', 1.0, 1)
```

Display the number of people with more than one pet with group by owner_id and having count of it is greater than 1

In [14]:
```
res = c.execute("SELECT count(*) FROM (SELECT count(owner_id) FROM pets GROUP BY ow
for row in res:
    print("{} People has more than one pet".format(row[0]))
```

```
43 People has more than one pet
```

6) How many pets have received treaments?

It can be fetched by treatments_done equal to 1

In [15]:
```
res = c.execute("SELECT count(*) FROM pets WHERE treatment_done=1")
for row in res:
    print(row)
```

```
(36,)
```

7) How many pets have received treatment that we know the type of?

It can be fetched by treatments_done equal to 1 and pet_type should not be null

In [16]:
```
res = c.execute("SELECT count(*) FROM pets WHERE treatment_done=1 AND pet_type IS N
for row in res:
    print(row)
```

```
(16,)
```

8) How many pets are there from the city called "east port"

Here the pets table is joined with persons table on id and person city is equal to `east port`

In [17]:
```
res = c.execute("SELECT count(*) FROM pets JOIN persons ON pets.owner_id = persons.
for row in res:
    print(row)
```

```
(49,)
```

9) How many pets are there from the city called "east port" and who received a treatment?

Here the pets table is joined with persons table on id and person city is equal to `east port` and pets `treatment_done` is equal to 1

In [18]:
```
res = c.execute("SELECT count(*) FROM pets JOIN persons ON pets.owner_id = persons.
for row in res:
    print(row)
```

```
(11,)
```