Week 1 Practice

You will need:

- Chapter 1 (SQL Cook Book). In this notebook you will be practicing the code provided in the chapter.
- Download emp.csv and dept.csv from the canvas Week 1 Practice

---

- Step 1-4: You will create a database week1.db
- Step 5: Practice Chapter 1 code
- Step 6: Close db connection
- Step 7: Open db connection using week1.db (you do not need step1-4 aanymore)

In [1]:
```
import sqlite3
import pandas as pd
```

STEP 1. Create a database named week1. You shsold have a new file week1.db in your local directory.

In [2]:
```
conn = sqlite3.connect('week1.db')
c = conn.cursor()
```

STEP 2. Read emp.csv and create a table emp

In [3]:
```
read_emp = pd.read_csv(r'emp.csv')
read_emp.to_sql('emp', conn, if_exists='append', index = False) # Insert the values from the csv file into the
```

STEP 3. read dept.csv and create a table dept

In [4]:
```
read_dept = pd.read_csv(r'dept.csv')
read_dept.to_sql('dept', conn, if_exists='append', index = False) # Insert the values from the csv file into th
```

Execution Examples

SQL statements will be executed with

c.execute(''' SQL code ''')

In [5]:
```
#Example 1
for row in c.execute('''
select * from emp
'''):
    print(row)
```

```
(7369, 'SMITH', 'CLERK', 7902.0, '17-Dec-05', 800, None, 20)
(7499, 'ALLEN', 'SALESMAN', 7698.0, '20-Feb-06', 1600, 300.0, 30)
(7521, 'WARD', 'SALESMAN', 7698.0, '22-Feb-06', 1250, 500.0, 30)
(7566, 'JONES', 'MANAGER', 7839.0, '2-Apr-06', 2975, None, 20)
(7654, 'MARTIN', 'SALESMAN', 7698.0, '28-Sep-06', 1250, 1400.0, 30)
(7698, 'BLAKE', 'MANAGER', 7839.0, '1-May-06', 2850, None, 30)
(7782, 'CLARK', 'MANAGER', 7839.0, '9-Jun-06', 2450, None, 10)
(7788, 'SCOTT', 'ANALYST', 7566.0, '9-Dec-07', 3000, None, 20)
(7839, 'KING', 'PRESIDENT', None, '17-Nov-06', 5000, None, 10)
(7844, 'TURNER', 'SALESMAN', 7698.0, '8-Sep-06', 1500, 0.0, 30)
(7876, 'ADAMS', 'CLERK', 7788.0, '12-Jan-08', 1100, None, 20)
(7900, 'JAMES', 'CLERK', 7698.0, '3-Dec-06', 950, None, 30)
(7902, 'FORD', 'ANALYST', 7566.0, '3-Dec-06', 3000, None, 20)
(7934, 'MILLER', 'CLERK', 7782.0, '23-Jan-07', 1300, None, 10)
```

In [6]:
```
colnames = c.description
for row in colnames:
    print(row[0])
```

```
EMPNO
ENAME
JOB
MGR
HIREDATE
SAL
COMM
DEPTNO
```

To print a table, use fetchall() to collect data and add column names thaht you have selected.

In [7]:
```
# Example 2
c.execute('''
select * from emp
''')

df = pd.DataFrame(c.fetchall(), columns=['EMPNO',
'ENAME',
'JOB',
'MGR',
'HIREDATE',
'SAL',
'COMM',
'DEPTNO'])
print(df)
```

```
    EMPNO   ENAME        JOB     MGR   HIREDATE   SAL    COMM  DEPTNO
0    7369   SMITH      CLERK  7902.0  17-Dec-05   800     NaN      20
1    7499   ALLEN   SALESMAN  7698.0  20-Feb-06  1600   300.0      30
2    7521    WARD   SALESMAN  7698.0  22-Feb-06  1250   500.0      30
3    7566   JONES    MANAGER  7839.0   2-Apr-06  2975     NaN      20
4    7654  MARTIN   SALESMAN  7698.0  28-Sep-06  1250  1400.0      30
5    7698   BLAKE    MANAGER  7839.0   1-May-06  2850     NaN      30
6    7782   CLARK    MANAGER  7839.0   9-Jun-06  2450     NaN      10
7    7788   SCOTT    ANALYST  7566.0   9-Dec-07  3000     NaN      20
8    7839    KING  PRESIDENT     NaN  17-Nov-06  5000     NaN      10
9    7844  TURNER   SALESMAN  7698.0   8-Sep-06  1500     0.0      30
10   7876   ADAMS    CLERK  7788.0  12-Jan-08  1100     NaN      20
11   7900   JAMES    CLERK  7698.0   3-Dec-06   950     NaN      30
12   7902    FORD   ANALYST  7566.0   3-Dec-06  3000     NaN      20
13   7934  MILLER    CLERK  7782.0  23-Jan-07  1300     NaN      10
```

Basics of SQL Queries

SELECT: Statement used to select rows and columns from a database.

FROM: Specifies which table in the database you want to direct your query to.

WHERE: Clause for filtering for specified value(s).

GROUP BY: Aggregating data. Needs to be used in conjunction with SQL aggregating functions like `SUM` and `COUNT`.

ORDER BY: Sorting columns in the database.

JOIN: Joins are used to combine tables with one another.

UNION, **INTERSECT/EXCEPT**: Set operations. Unioning in SQL allows one to append tables on top of one another.

Step 5. Practice Chapter 1

In [8]:
```
## Your turn
```

Step 6. Close the connection

In [9]:
```
conn.close()
```

Step 7. Open connection with your database week1.db

In [10]:
```
conn = sqlite3.connect('week1.db')
c = conn.cursor()
```

In [11]:
```
## You can continue working with SQL coding now
```

In [17]:
```
for row in c.description:
    print(row[0])
```

```
EMPNO
ENAME
JOB
MGR
HIREDATE
SAL
COMM
DEPTNO
```

Question 1.1: You have a table and want to see all of the data in it.

In [12]:
```
for row in c.execute('''select * from emp'''):
    print(row)
```

```
(7369, 'SMITH', 'CLERK', 7902.0, '17-Dec-05', 800, None, 20)
(7499, 'ALLEN', 'SALESMAN', 7698.0, '20-Feb-06', 1600, 300.0, 30)
(7521, 'WARD', 'SALESMAN', 7698.0, '22-Feb-06', 1250, 500.0, 30)
(7566, 'JONES', 'MANAGER', 7839.0, '2-Apr-06', 2975, None, 20)
(7654, 'MARTIN', 'SALESMAN', 7698.0, '28-Sep-06', 1250, 1400.0, 30)
(7698, 'BLAKE', 'MANAGER', 7839.0, '1-May-06', 2850, None, 30)
(7782, 'CLARK', 'MANAGER', 7839.0, '9-Jun-06', 2450, None, 10)
(7788, 'SCOTT', 'ANALYST', 7566.0, '9-Dec-07', 3000, None, 20)
(7839, 'KING', 'PRESIDENT', None, '17-Nov-06', 5000, None, 10)
(7844, 'TURNER', 'SALESMAN', 7698.0, '8-Sep-06', 1500, 0.0, 30)
(7876, 'ADAMS', 'CLERK', 7788.0, '12-Jan-08', 1100, None, 20)
(7900, 'JAMES', 'CLERK', 7698.0, '3-Dec-06', 950, None, 30)
(7902, 'FORD', 'ANALYST', 7566.0, '3-Dec-06', 3000, None, 20)
(7934, 'MILLER', 'CLERK', 7782.0, '23-Jan-07', 1300, None, 10)
```

Question 1.2: You have a table and want to see only rows that satisfy a specific condition.

In [18]:
```
for row in c.execute('''select * from emp where deptno=20'''):
    print(row)
```

```
(7369, 'SMITH', 'CLERK', 7902.0, '17-Dec-05', 800, None, 20)
(7566, 'JONES', 'MANAGER', 7839.0, '2-Apr-06', 2975, None, 20)
(7788, 'SCOTT', 'ANALYST', 7566.0, '9-Dec-07', 3000, None, 20)
(7876, 'ADAMS', 'CLERK', 7788.0, '12-Jan-08', 1100, None, 20)
(7902, 'FORD', 'ANALYST', 7566.0, '3-Dec-06', 3000, None, 20)
```

Question 1.3: You want to return rows that satisfy multiple conditions.

In [27]:
```
for row in c.execute('''select * from emp where ((deptno=20
                        and comm is not null)
                        or sal <= 2000) and job = "CLERK" '''):
    print(row)
```

```
(7369, 'SMITH', 'CLERK', 7902.0, '17-Dec-05', 800, None, 20)
(7876, 'ADAMS', 'CLERK', 7788.0, '12-Jan-08', 1100, None, 20)
(7900, 'JAMES', 'CLERK', 7698.0, '3-Dec-06', 950, None, 30)
(7934, 'MILLER', 'CLERK', 7782.0, '23-Jan-07', 1300, None, 10)
```

Return specific colulmns

Question 1.4: You have a table and want to see values for specific columns rather than for all the columns.

In [28]:
```
for row in c.execute('''SELECT ename, deptno, sal, Job from emp'''):
    print(row)
```

```
('SMITH', 20, 800, 'CLERK')
('ALLEN', 30, 1600, 'SALESMAN')
('WARD', 30, 1250, 'SALESMAN')
('JONES', 20, 2975, 'MANAGER')
('MARTIN', 30, 1250, 'SALESMAN')
('BLAKE', 30, 2850, 'MANAGER')
('CLARK', 10, 2450, 'MANAGER')
('SCOTT', 20, 3000, 'ANALYST')
('KING', 10, 5000, 'PRESIDENT')
('TURNER', 30, 1500, 'SALESMAN')
('ADAMS', 20, 1100, 'CLERK')
('JAMES', 30, 950, 'CLERK')
('FORD', 20, 3000, 'ANALYST')
('MILLER', 10, 1300, 'CLERK')
```

Question 1.5: You would like to change the names of the columns that are returned by your query so they are more readable and understandable. Consider this query that returns the salaries and commissions for each employee:

1 select sal,comm
2 from emp
What's SAL? Is it short for sale? Is it someone's name? What's COMM? Is it communi- cation? You want the results to have more meaningful labels.

To change the names of the columns using AS keyword. This is known as aliasing.

In [13]:
```
c.execute('''SELECT sal AS salary, comm AS commission FROM emp''')

df = pd.DataFrame(c.fetchall(), columns=['SALARY', 'COMMISSION'])
print(df)
```

```
    SALARY  COMMISSION
0      800         NaN
1     1600       300.0
2     1250       500.0
3     2975         NaN
4     1250      1400.0
5     2850         NaN
6     2450         NaN
7     3000         NaN
8     5000         NaN
9     1500         0.0
10    1100         NaN
11     950         NaN
12    3000         NaN
13    1300         NaN
```