



Introduction to database with Python

BY: HOSSEIN HAGHBIN

Basics about a database?

- A **database** is a structured collection of records.
- Database Management System (**DBMS**)
 - add, remove, update records
 - retrieve data that match certain criteria
 - cross-reference data in different tables
 - perform complex aggregate calculation
- Database consists of columns (attributes) and rows (records).
- Databases versus spreadsheets
 - easy manipulation of data

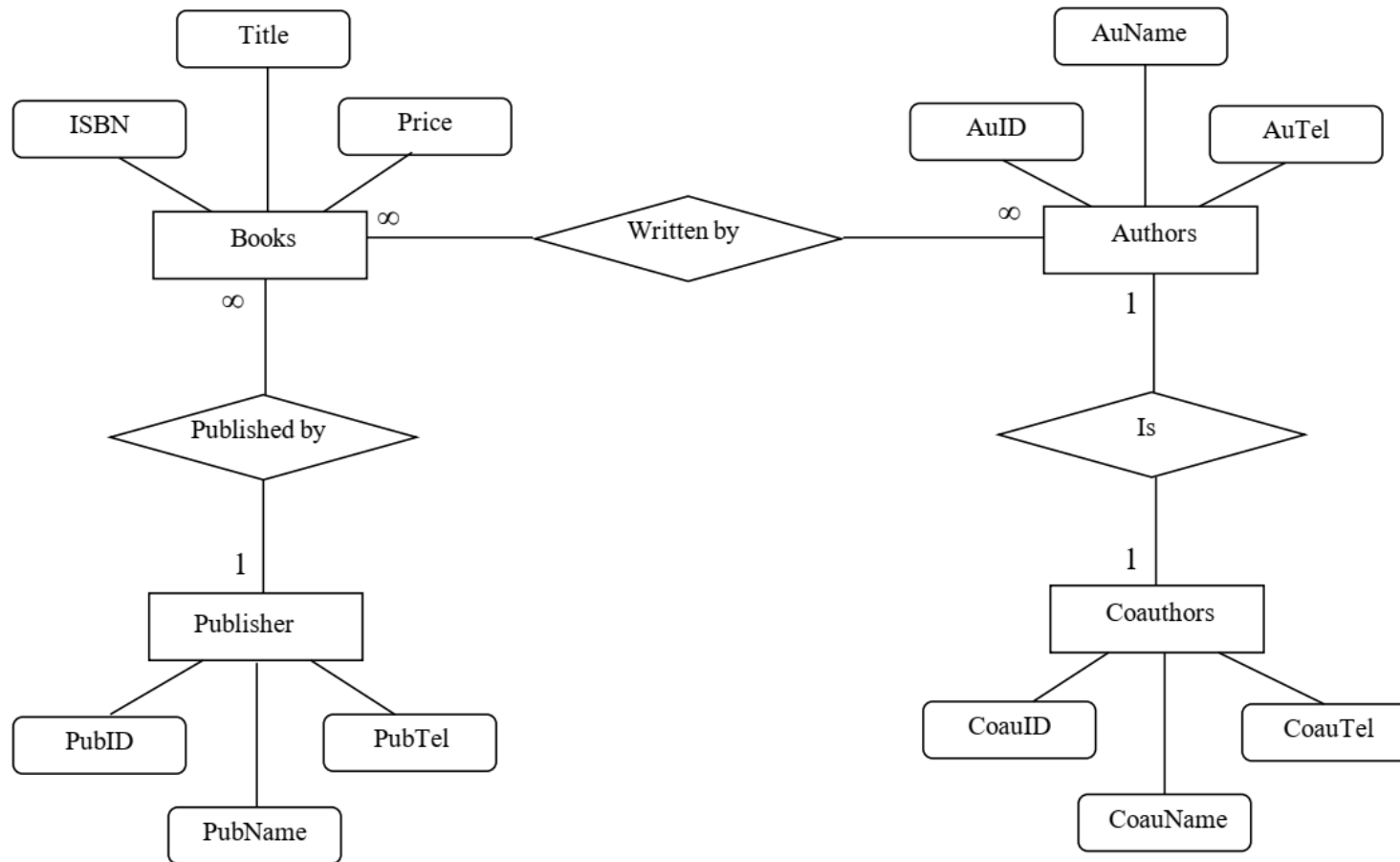


Single table database

ISBN	Title	AuID	AuName	AuTel	PubID	PubName	PubTel	Price
0-99-999999-9	Emma	1	Austen	111-111-1111	1	Big House	123-456-7890	20.00 zł
0-91-335678-7	Faerie Queen	7	Spenser	777-777-7777	1	Big House	123-456-7890	17.00 zł
0-91-045678-5	Hamlet	5	Shakespeare	555-555-5555	2	Alpha Press	999-999-9999	20.00 zł
0-103-45678-9	Iliad	3	Homer	333-333-3333	1	Big House	123-456-7890	25.00 zł
0-555-55555-9	Macbeth	5	Shakespeare	555-555-5555	2	Alpha Press	999-999-9999	12.00 zł
0-55-123456-9	Main Street	10	Jones	123-333-3333	3	Small House	714-000-0000	23.00 zł
0-55-123456-9	Main Street	9	Smith	123-222-2222	3	Small House	714-000-0000	23.00 zł
0-12-333433-3	On Liberty	8	Mill	888-888-8888	1	Big House	123-456-7890	25.00 zł
0-321-32132-1	Balloon	2	Sleepy	222-222-2222	3	Small House	714-000-0000	34.00 zł
0-321-32132-1	Balloon	4	Snoopy	444-444-4444	3	Small House	714-000-0000	34.00 zł
0-321-32132-1	Balloon	11	Grumpy	321-321-0000	3	Small House	714-000-0000	34.00 zł



Relational Database: system of related tables



Home library – table Books

ISBN	Title	PubID	Price
0-103-45678-9	Iliad	1	25.00 zł
0-11-345678-9	Moby Dick	3	49.00 zł
0-12-333433-3	On Liberty	1	25.00 zł
0-123-45678-0	Ulysses	2	34.00 zł
0-12-345678-9	Jane Eyre	3	49.00 zł
0-321-32132-1	Balloon	3	34.00 zł
0-55-123456-9	Main Street	3	23.00 zł
0-555-55555-9	Macbeth	2	12.00 zł
0-91-045678-5	Hamlet	2	20.00 zł
0-91-335678-7	Faerie Queen	1	15.00 zł
0-99-777777-7	King Lear	2	49.00 zł
0-99-999999-9	Emma	1	20.00 zł
1-1111-1111-1	C++	1	30.00 zł
1-22-233700-0	Visual Basic	1	25.00 zł



Home library – table Authors

AuID	AuName	AuTel
1	Austen	111-111-1111
2	Melville	222-222-2222
3	Homer	333-333-3333
4	Roman	444-444-4444
5	Shakespeare	555-555-5555
6	Joyce	666-666-6666
7	Spenser	777-777-7777
8	Mill	888-888-8888
9	Smith	123-222-2222
10	Jones	123-333-3333
11	Snoopy	321-321-2222
12	Grumpy	321-321-0000
13	Sleepy	321-321-1111



Home library – table Publishers

PubID	PubName	PubTel
1	Big House	123-456-7890
2	Alpha Press	999-999-9999
3	Small House	714-000-0000



Home library – table Books/Authors

ISBN	AuID
0-103-45678-9	3
0-11-345678-9	2
0-12-333433-3	8
0-123-45678-0	6
0-12-345678-9	1
0-321-32132-1	11
0-321-32132-1	12
0-321-32132-1	13
0-55-123456-9	9
0-55-123456-9	10
0-555-55555-9	5
0-91-045678-5	5
0-91-335678-7	7
0-99-777777-7	5
0-99-999999-9	1
1-1111-1111-1	4
1-22-233700-0	4



Table

- Unique name
- Size = # of rows, order =# of columns
- Structure of a table $\rightarrow T\{ A_1, A_2, \dots, A_n \}$
- All rows different
- Order of rows not important
- Unique headers identify columns
- NULL value in tables



Database keys

- **Primary key**

- Value unique for each record in a table
- This value can not be used twice
- AutoNumber guarantees uniqueness but does not carry any useful information

- **Foreign keys**

- Used to create relationships between tables
- No uniqueness constraint for foreign keys



Queries

- **Database** = data located in tables + relations
- **Query** = primary mechanism for retrieving information from a database, consists of questions presented to the database in a predefined format = an expression stored in a database having a unique name
- **Answer to the query** = a computed table



SQL

SQL is a database computer language designed for the retrieval and management of data in a relational database. **SQL** stands for **Structured Query Language**. All the Relational Database Management Systems (**RDMS**) like MySQL, MS Access, Oracle, SQLite, Informix, Postgres and SQL Server use SQL as their standard database language.



SQLite

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world. The source code for SQLite is in the public domain.



Python SQLite

SQLite3 can be integrated with Python using sqlite3 module, which was written by Gerhard Haring. It provides an SQL interface compliant with the DB-API 2.0 specification described by PEP 249. You do not need to install this module separately because it is shipped by default along with Python version 2.5.x onwards.



Python SQLite

To use sqlite3 module, you must:

- First create a connection object that represents the database
- Optionally, you can create a cursor object, which will help you in executing all the SQL statements.



Establishing Connection

To establish a connection with SQLite3 database using python you need to:

- Import the sqlite3 module using the import statement.
- The connect() method accepts the name of the database you need to connect with as a parameter and, returns a Connection object.



Example :

```
import sqlite3
conn = sqlite3.connect('example.db')
print("Connection established .....")
```

Output:

```
Connection established .....
```



Create Table

Following is the syntax to create a table in SQLite database:

```
CREATE TABLE database_name.table_name(  
    column1 datatype PRIMARY KEY(one or more columns),  
    column2 datatype,  
    column3 datatype,  
    .....  
    columnN datatype  
);
```



Example :Following Python program creates a table named Employee in SQLite3

```
import sqlite3
#Connecting to sqlite
conn = sqlite3.connect('example.db')
#Creating a cursor object using the cursor() method
cursor = conn.cursor()
#Doping EMPLOYEE table if already exists.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")
```



Example :Following Python program creates a table named Employee in SQLite3

```
#Creating table as per requirement
```

```
sql = '''CREATE TABLE EMPLOYEE(  
    FIRST_NAME CHAR(20) NOT NULL,  
    LAST_NAME CHAR(20),  
    AGE INT,  
    SEX CHAR(1),  
    INCOME FLOAT) '''
```

```
cursor.execute(sql)
```



Example :Following Python program creates a table named Employee in SQLite3

```
print("Table created successfully.....")

# Commit your changes in the database
conn.commit()

#Closing the connection
conn.close()
```



Insert Data

You can add new rows to an existing table of SQLite using the INSERT INTO statement. You can add new rows to an existing table of SQLite using the INSERT INTO statement.

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);
```



Example :Following python example inserts records into to a table named EMPLOYEE:

```
import sqlite3

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

q1 = '''INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX, INCOME)
      VALUES ('Fatemeh', 'Rahmani', 27, 'F', 9000) '''
q2 = '''INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX, INCOME)
      VALUES ('Vahid', 'Bahmabi', 20, 'M', 6000)'''
q3 = '''INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX, INCOME)
      VALUES ('Shahin', 'Sheikhi', 25, 'M', 8300 )'''
q4 = '''INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX, INCOME)
      VALUES ('Sarmira', 'Sharmsar', 26, 'F', 10000)'''
q5 = '''INSERT INTO EMPLOYEE(FIRST_NAME, LAST_NAME, AGE, SEX, INCOME)
      VALUES ('Tahereh', 'Mishekari', 24, 'F', 6000) '''
```

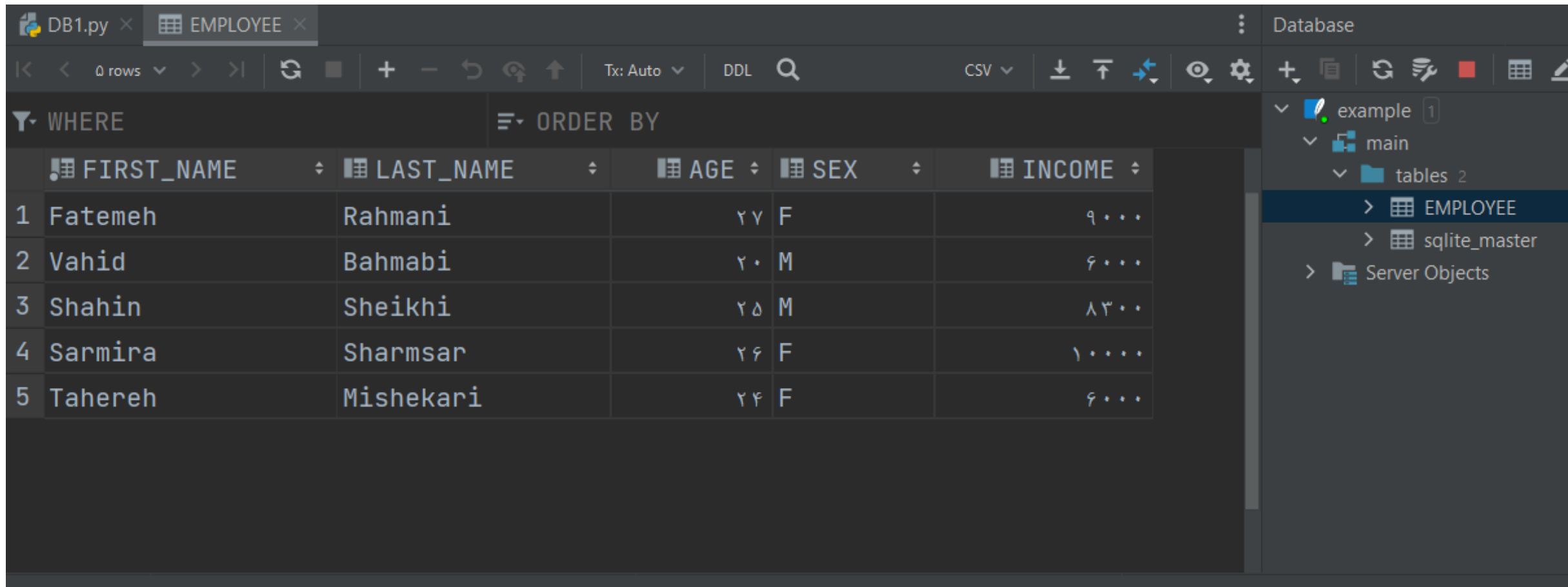


Example :Following python example inserts records into to a table named EMPLOYEE:

```
cursor.execute(q1)
cursor.execute(q2)
cursor.execute(q3)
cursor.execute(q4)
cursor.execute(q5)
conn.commit()
conn.close()
```



Example :Following python example inserts records into to a table named EMPLOYEE:



The screenshot shows a database management interface with a table named 'EMPLOYEE'. The table has five columns: FIRST_NAME, LAST_NAME, AGE, SEX, and INCOME. Five records are displayed, numbered 1 to 5. The interface includes a toolbar with various icons for navigation and editing, and a sidebar on the right showing the database structure with 'example' as the selected database.

	FIRST_NAME	LAST_NAME	AGE	SEX	INCOME
1	Fatemeh	Rahmani	۲۷	F	۹۰۰۰
2	Vahid	Bahmabi	۲۰	M	۶۰۰۰
3	Shahin	Sheikhi	۲۵	M	۸۳۰۰
4	Sarmira	Sharmsar	۲۶	F	۱۰۰۰۰
5	Tahereh	Mishekari	۲۴	F	۶۰۰۰

Select Data

Following is the syntax of the SELECT statement in SQLite:

```
SELECT column1, column2, columnN FROM table_name;
```

The **sqlite3.Cursor** class provides three methods namely **fetchall()**, **fetchmany()** and, **fetchone()** where,

- The **fetchall()** method retrieves all the rows in the result set of a query and returns them as list of tuples. (If we execute this after retrieving few rows it returns the remaining ones).
- The **fetchone()** method fetches the next row in the result of a query and returns it as a tuple.
- The **fetchmany()** method is similar to the **fetchone()** but, it retrieves the next set of rows in the result set of a query, instead of a single row.



Example :Following example fetches all the rows of the EMPLOYEE table using the SELECT query:

```
import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
cursor.execute(''SELECT * from EMPLOYEE'')
result = cursor.fetchone();
print(result)
#Fetching 1st row from the table
result = cursor.fetchall();
print(result)
conn.commit()
conn.close()
```



Example :Following example fetches all the rows of the EMPLOYEE table using the SELECT query:

Output:

```
('Fatemeh', 'Rahmani', 27, 'F', 9000.0)
```

```
[('Vahid', 'Bahmabi', 20, 'M', 6000.0), ('Shahin', 'Sheikhi', 25, 'M',  
8300.0), ('Sarmira', 'Sharmsar', 26, 'F', 10000.0), ('Tahereh',  
'Mishekari', 24, 'F', 6000.0)]
```



Where Clause

If you want to **fetch**, **delete** or, **update** particular rows of a table in SQLite, you need to use the where clause to specify condition to filter the rows of the table for the operation. For example, if you have a **SELECT** statement with where clause, only the rows which satisfies the specified condition will be retrieved.



Where Clause Syntax

Following is the syntax of the WHERE clause in SQLite:

```
SELECT column1, column2, columnN  
  
FROM table_name  
  
WHERE [search_condition]
```

You can specify a *search_condition* using comparison or logical operators. Like

>, <, =, LIKE, NOT, etc.



Example :Following example creates a table named Employee and populates it. Then using the where clause it retrieves the records with age value less than 23

```
import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
cursor.execute("SELECT * from EMPLOYEE WHERE AGE <23")
print(cursor.fetchall())
conn.commit()
conn.close()
```

Output

```
[('Vahid', 'Bahmabi', 20, 'M', 6000.0)]
```



Order By

While fetching data using **SELECT** query, you will get the records in the same order in which you have inserted them.

You can sort the results in desired order (ascending or descending) using the **Order By** clause. By default, this clause sorts results in ascending order, if you need to arrange them in descending order you need to use “**DESC**” explicitly.



Order By Syntax

Following is the syntax of the ORDER BY clause in SQLite.

```
SELECT column-list  
FROM table_name  
[WHERE condition]  
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```



Example :Following example creates a table named Employee and populates it. Then using the where clause it retrieves the records with age value less than 23

```
import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
#Retrieving specific records using the ORDER BY clause
cursor.execute("SELECT * from EMPLOYEE ORDER BY AGE")
print(cursor.fetchall())
conn.close()
```

Output:

```
[('Vahid', 'Bahmabi', 20, 'M', 6000.0),
 ('Tahereh', 'Mishekari', 24, 'F', 6000.0),
 ('Shahin', 'Sheikhi', 25, 'M', 8300.0),
 ('Sarmira', 'Sharmsar', 26, 'F', 10000.0),
 ('Fatemeh', 'Rahmani', 27, 'F', 9000.0)]
```

Update Table

UPDATE Operation on any database implies modifying the values of one or more records of a table, which are already available in the database. Following is the syntax of the UPDATE statement in SQLite:

```
UPDATE table_name  
SET column1 = value1, column2 = value2....., columnN = valueN  
WHERE [condition];
```

Example :Following Python example, creates a table with name EMPLOYEE, inserts 5 records into it and, increases the age of all the male employees by 1:

```
import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
#Fetching all the rows before the update
print("Contents of the Employee table: ")
cursor.execute('''SELECT * from EMPLOYEE''')
print(cursor.fetchall())
#Updating the records
sql = '''UPDATE EMPLOYEE SET AGE=AGE+1 WHERE SEX = 'M' '''
cursor.execute(sql)
print("Contents of the Employee table after the update operation: ")
cursor.execute('''SELECT * from EMPLOYEE''')
print(cursor.fetchall())
#Commit your changes in the database
conn.commit()
conn.close()
```

Delete Data

To delete records from a SQLite table, you need to use the DELETE FROM statement. To remove specific records, you need to use WHERE clause along with it. Following is the syntax of the DELETE query in SQLite:

```
DELETE FROM table_name [WHERE Clause]
```



Example :Following python example deletes the records from EMPLOYEE table with age value greater than 25.

```
import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
cursor.execute('''SELECT * from EMPLOYEE''')
print(cursor.fetchall())
#Deleting records
cursor.execute('''DELETE FROM EMPLOYEE WHERE AGE > 25''')
#Retrieving data after delete
print("Contents of the table after delete operation ")
cursor.execute("SELECT * from EMPLOYEE")
print(cursor.fetchall())
#Commit your changes in the database
conn.commit()
conn.close()
```

Join

When you have divided the data in two tables you can fetch combined records from these two tables using Joins.



Example :Following SQLite example, demonstrates the JOIN clause using python:

```
import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
cursor.execute("DROP TABLE IF EXISTS Score")
sql = '''CREATE TABLE Score(
        FIRST_NAME CHAR(20) NOT NULL,
        LAST_NAME CHAR(20),
        Grade INT)'''
q1 = '''INSERT INTO Score(FIRST_NAME, LAST_NAME, Grade)
        VALUES ('Fatemeh', 'Rahmani', 15) '''
q2 = '''INSERT INTO Score(FIRST_NAME, LAST_NAME, Grade)
        VALUES ('Vahid', 'Bahmabi', 18)'''
q3 = '''INSERT INTO Score(FIRST_NAME, LAST_NAME, Grade)
        VALUES ('Shahin', 'Sheikhi', 19)'''
q4 = '''INSERT INTO Score(FIRST_NAME, LAST_NAME, Grade)
        VALUES ('Sarmira', 'Sharmsar', 16)'''
```


Example :Following SQLite example, demonstrates the JOIN clause using python:

```
cursor.execute(sql)
cursor.execute(q1)
cursor.execute(q2)
cursor.execute(q3)
cursor.execute(q4)
conn.commit()

sql = '''SELECT * from EMPLOYEE
        JOIN Score ON EMPLOYEE.LAST_NAME=Score.LAST_NAME'''
cursor.execute(sql)
result = cursor.fetchall();
print(result)
conn.commit()
conn.close()
```

نکات تکمیلی SQLite

برای دستور INSERT می توان به صورت زیر نیز عمل کرد:

```
import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
q1 = '''INSERT INTO EMPLOYEE VALUES ('Ali', 'Rezaie', 27, 'M', 9000) '''
q2 = '''INSERT INTO EMPLOYEE VALUES ('Zahra', 'Rezaie', ?, 'F', ?) '''
cursor.execute(q1)
cursor.execute(q2, (23, 8500))
conn.commit()
conn.close()
```



نکات تکمیلی SQLite

برای دستور INSERT می توان به صورت زیر نیز عمل کرد:

```
def insert(name, family, age, gender, inc):  
    conn = sqlite3.connect('example.db')  
    cur = conn.cursor()  
    q = "INSERT INTO EMPLOYEE VALUES (?, ?, ?, ?, ?) "  
    name = name.capitalize()  
    family = family.capitalize()  
    cur.execute(q, (name, family, age, gender, inc))  
    out = cur.fetchall()  
    conn.commit()  
    conn.close()  
    print('Data inserted successfully ...')  
  
insert('Ahmad', 'Rahimi', 27, 'M', 12000)
```



نکات تکمیلی SQLite

برای دستور SELECT می توان به صورت زیر نیز عمل کرد:

```
import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
q1 = '''SELECT * from EMPLOYEE WHERE AGE>? '''
q2 = '''SELECT * from EMPLOYEE WHERE AGE>? OR INCOME>? '''
cursor.execute(q1,(20,))
cursor.fetchall()
cursor.execute(q2,(26,8500))
cursor.fetchall()
```



نکات تکمیلی SQLite

برای دستور SELECT می توان به صورت زیر نیز عمل کرد:

```
def search(name,family):  
    conn = sqlite3.connect('example.db')  
    cur = conn.cursor()  
    q = "SELECT * FROM EMPLOYEE WHERE FIRST_NAME=? and LAST_NAME=?"  
    name = name.capitalize()  
    family = family.capitalize()  
    cur.execute(q,(name,family))  
    out = cur.fetchall()  
    return out  
  
search('Tahereh','mishekari')
```



نکات تکمیلی SQLite

برای دستور SELECT می توان به صورت زیر نیز عمل کرد:

```
def view():  
    connection = sqlite3.connect("example.db")  
    cur = connection.cursor()  
    q1 = """  
    SELECT * FROM EMPLOYEE  
    """  
  
    cur.execute(q1)  
    rows = cur.fetchall()  
    connection.commit()  
    connection.close()  
    return rows
```



تعریف نام کاربری و رمز عبور در SQLite

```
import sqlite3
conn = sqlite3.connect('example.db')
cur = conn.cursor()
cur.execute("DROP TABLE IF EXISTS users")
q1=''
CREATE TABLE users (
    ID INTEGER PRIMARY KEY AUTOINCREMENT,
    username CHAR(20) NOT NULL,
    password CHAR(20) NOT NULL)
...
q2 = "INSERT INTO users VALUES (NULL,'admin','admin')"
cur.execute(q1)
cur.execute(q2)
conn.commit()
conn.close()
```



تعریف نام کاربری و رمز عبور در SQLite

```
def connect(username,password):  
    conn = sqlite3.connect("example.db")  
    cur = conn.cursor()  
    q = f"SELECT username from users WHERE username='{username}' AND Password =  
        '{password}';"  
    cur.execute(q)  
    if not cur.fetchone(): # An empty result evaluates to False.  
        print("Login failed...")  
        conn.close()  
    else:  
        print("Welcome...")  
  
connect('boss','1234')
```