SE-019 Advanced Programming

Tauseef Ahmed, PhD
Tauseef.Ahmed@eek.ee

Lecture 6

- Charting Data
- Python Image Library
- Databases and SQL

CHARTING DATA

Data Visualization

- Beautiful charts that visually represent data
- Depending on the format of the data source, we can plot one or more columns of data in the same chart.
- We will be using the Python **Matplotlib** module to create our charts.
- Matplotlib Python module, which enables us to create visual charts using Python

Matplotlib

- **`matplotlib.pyplot`** is a collection of command style functions that make **`matplotlib`** work like MATLAB
- Each **`pyplot`** function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.
- In **`matplotlib.pyplot`** various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes

matplotlib Installation

```
python -m pip install -U pip
python -m pip install -U matplotlib
```

- For me this always have worked for various third party packages

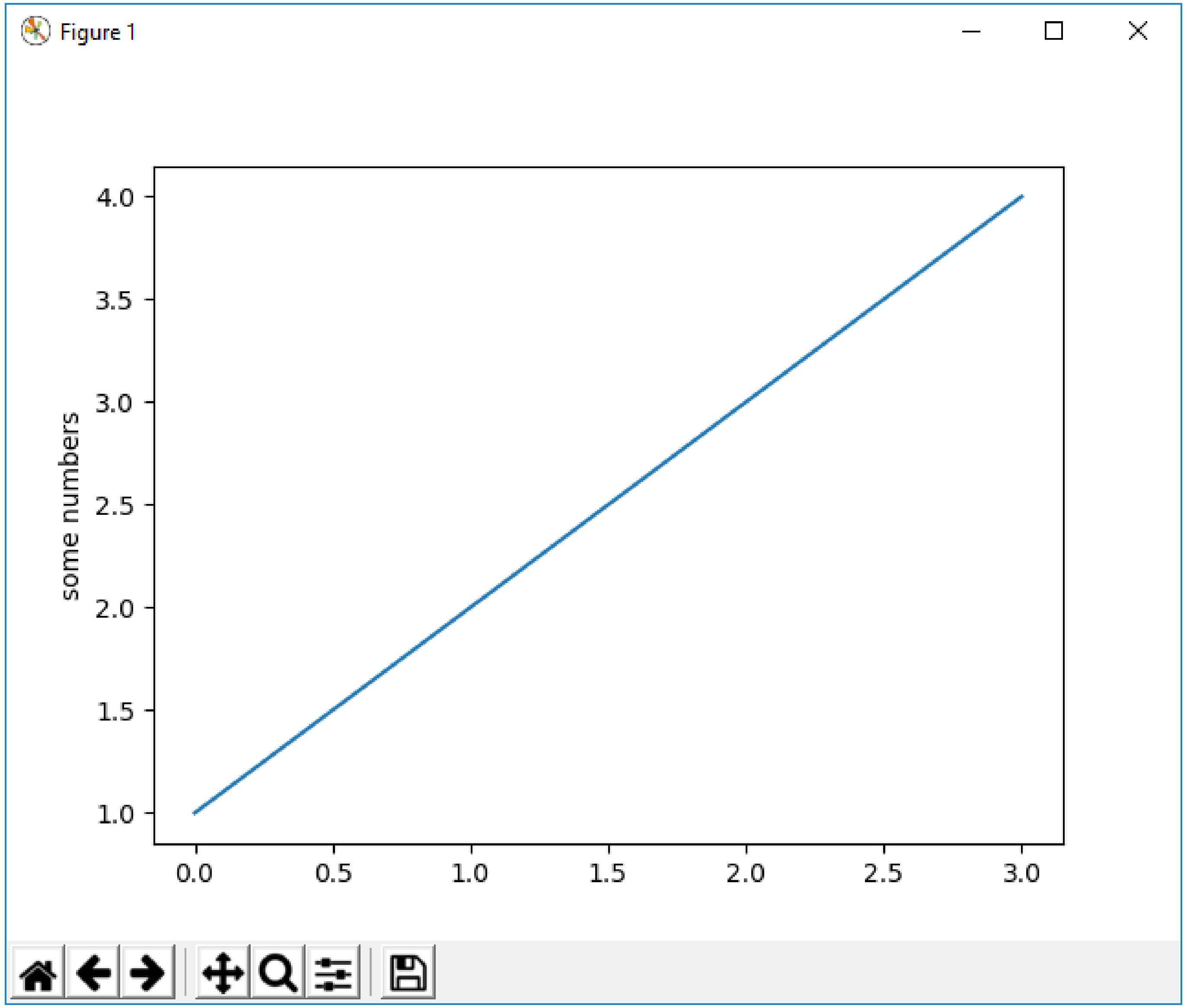
```
pip3 install matplotlib
```

https://matplotlib.org/stable/

Example

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```

Example



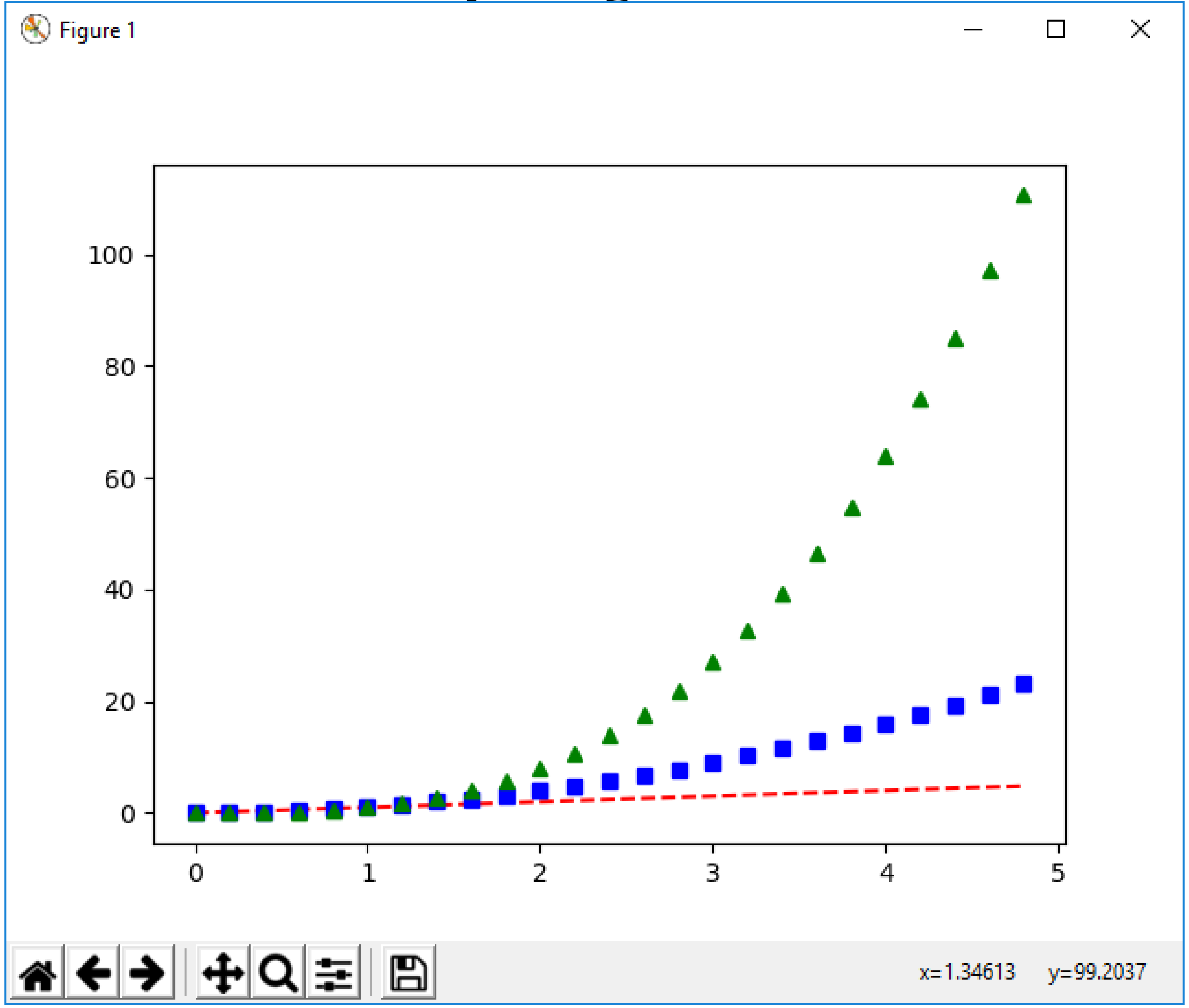
Comparison Data

```
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0.0, 5.0, 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

Comparing Data



More than one plot in a single window

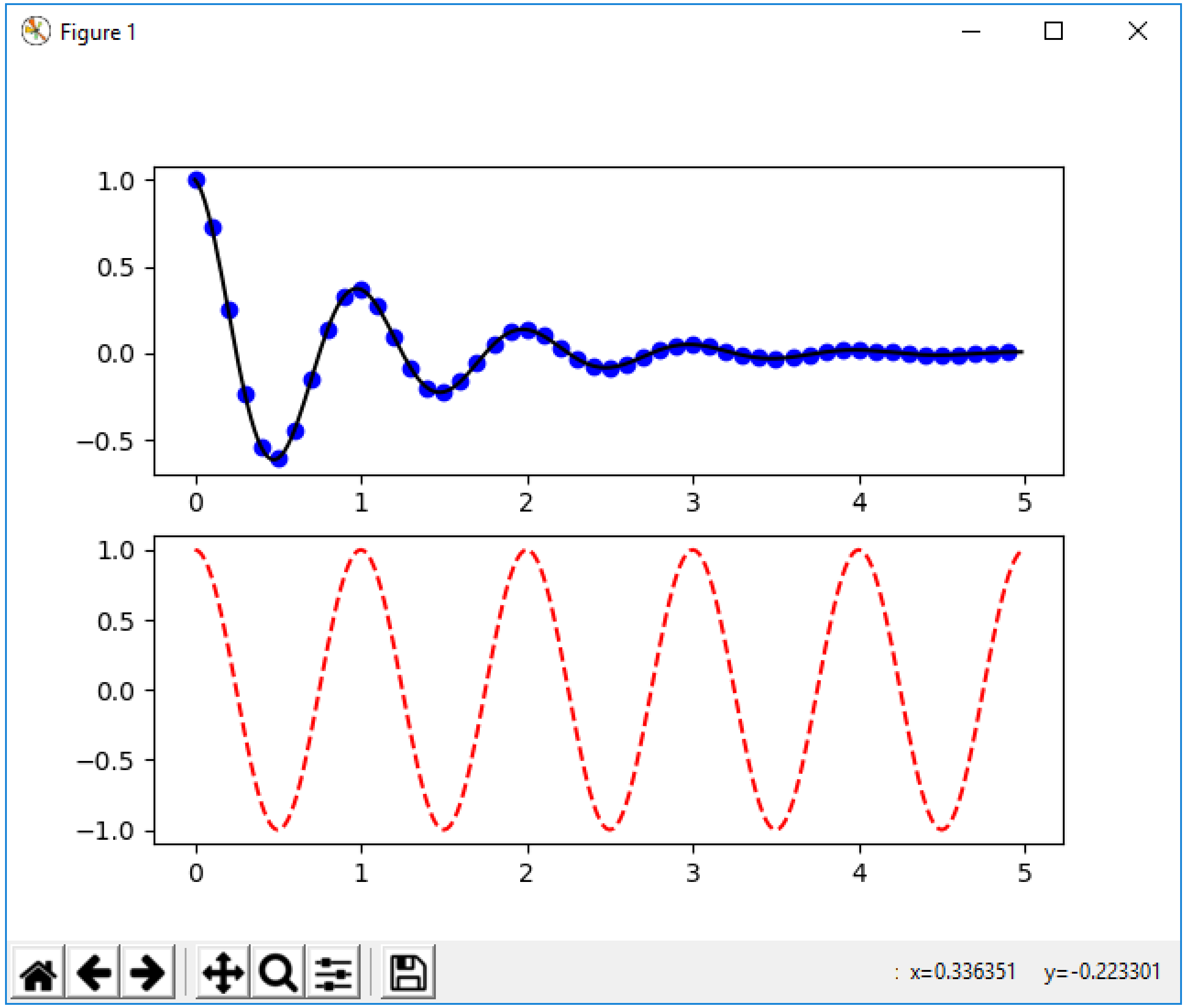
```
import numpy as np
import matplotlib.pyplot as plt

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```



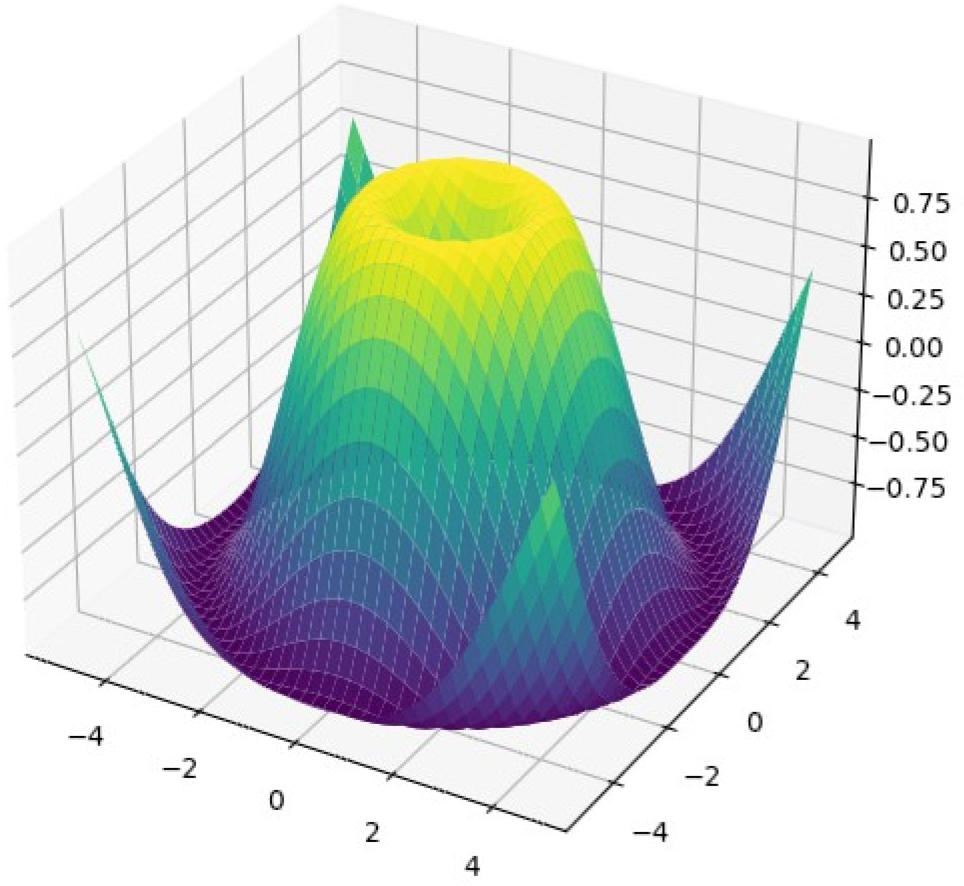
3d Surface plot

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

fig = plt.figure()
ax = Axes3D(fig, auto_add_to_figure=False)
fig.add_axes(ax)
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.viridis)

plt.show()
```



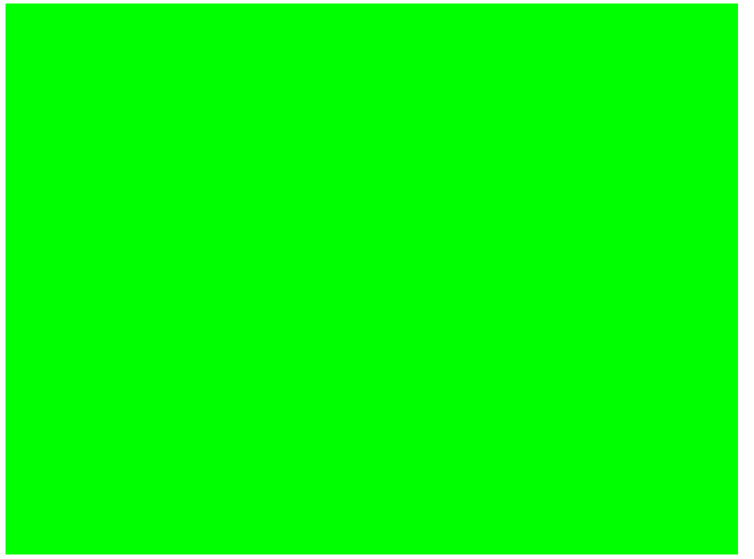
PYTHON IMAGING LIBRARY

Python Imaging Library

- Python imaging library (Pillow)
- First, download if necessary
  - `pip install pillow`
- Defines module `Image` and many useful functions
- Online Help
  - http://pillow.readthedocs.io/en/5.1.x/index.html#
  - https://python-pillow.org/

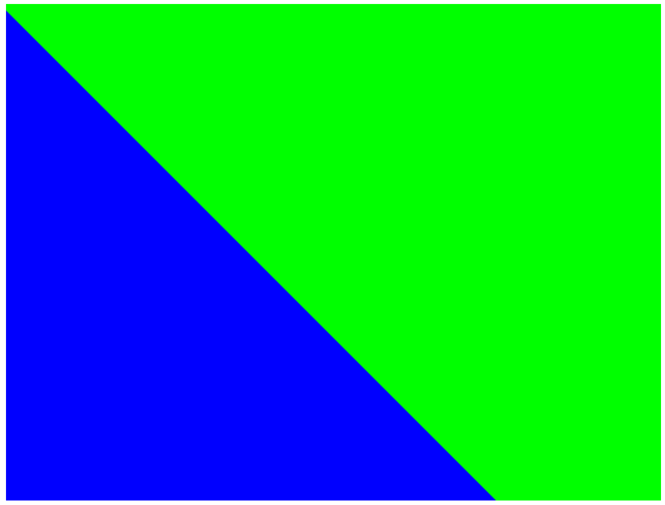
Creating Image

```
from PIL import Image
img=Image.new('RGB', (640,480), (0,255,0))
img.save('green.png')
```



Creating Image

```
from PIL import Image
img=Image.new('RGB',(640,480),(0,0,255))
yp = 0
while yp < 480:
   xp = 0
   while xp < 640:
      if xp>yp:
         img.putpixel((xp,yp),(0,255,0))
      xp += 1
   yp += 1
img.show()
```



Creating Image

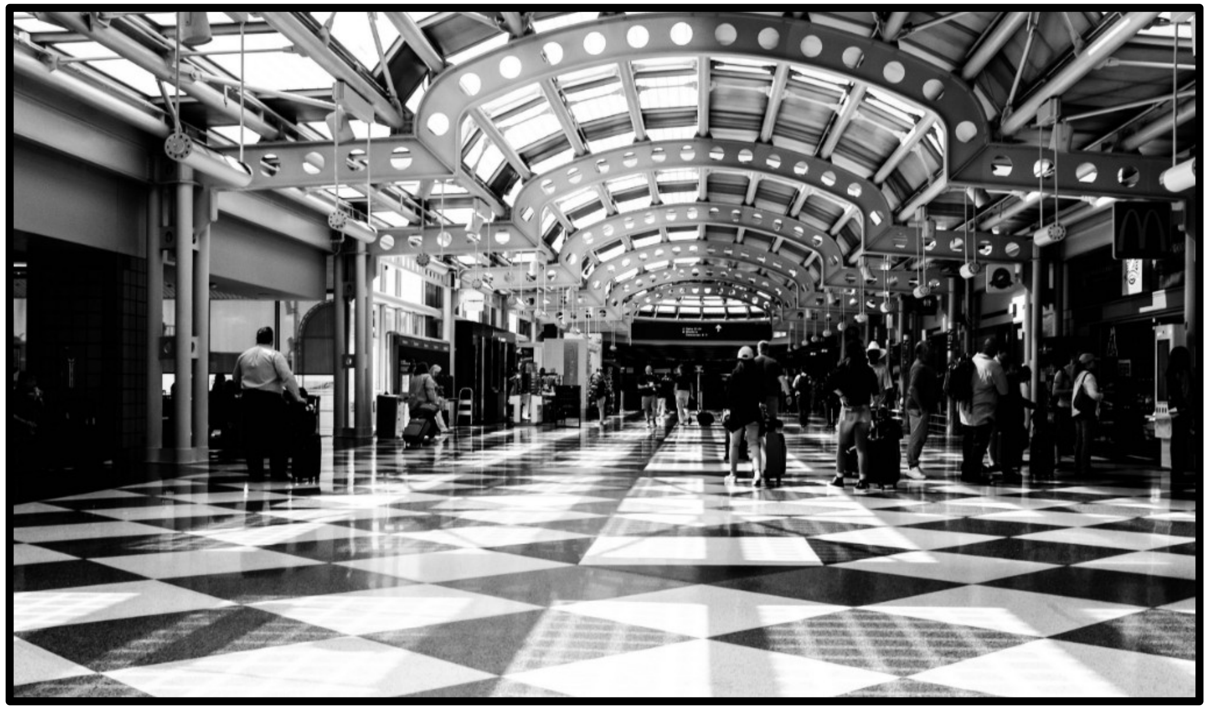
- `PIL.Image.new(mode, size, color)`
  - Creates a new image with the given mode and size.
- `PIL.Image.fromarray(obj, mode = None)`
  - Creates an image memory from an object exporting the array interface (using the buffer protocol).
- `PIL.Image.frombytes(mode, size, data, decoder_name='raw', *args)`
  - Creates a copy of an image memory from pixel data in a buffer.
- `PIL.Image.frombuffer(mode, size, data, decoder_name='raw', *args)`
  - Creates an image memory referencing pixel data in a byte buffer.

Opening Image

- `PIL.Image.open(fp, mode='r')`
  - Opens given image file. If given, the mode argument must be “r”.
  - `IOError` – If the file cannot be found, or the image cannot be opened and identified.

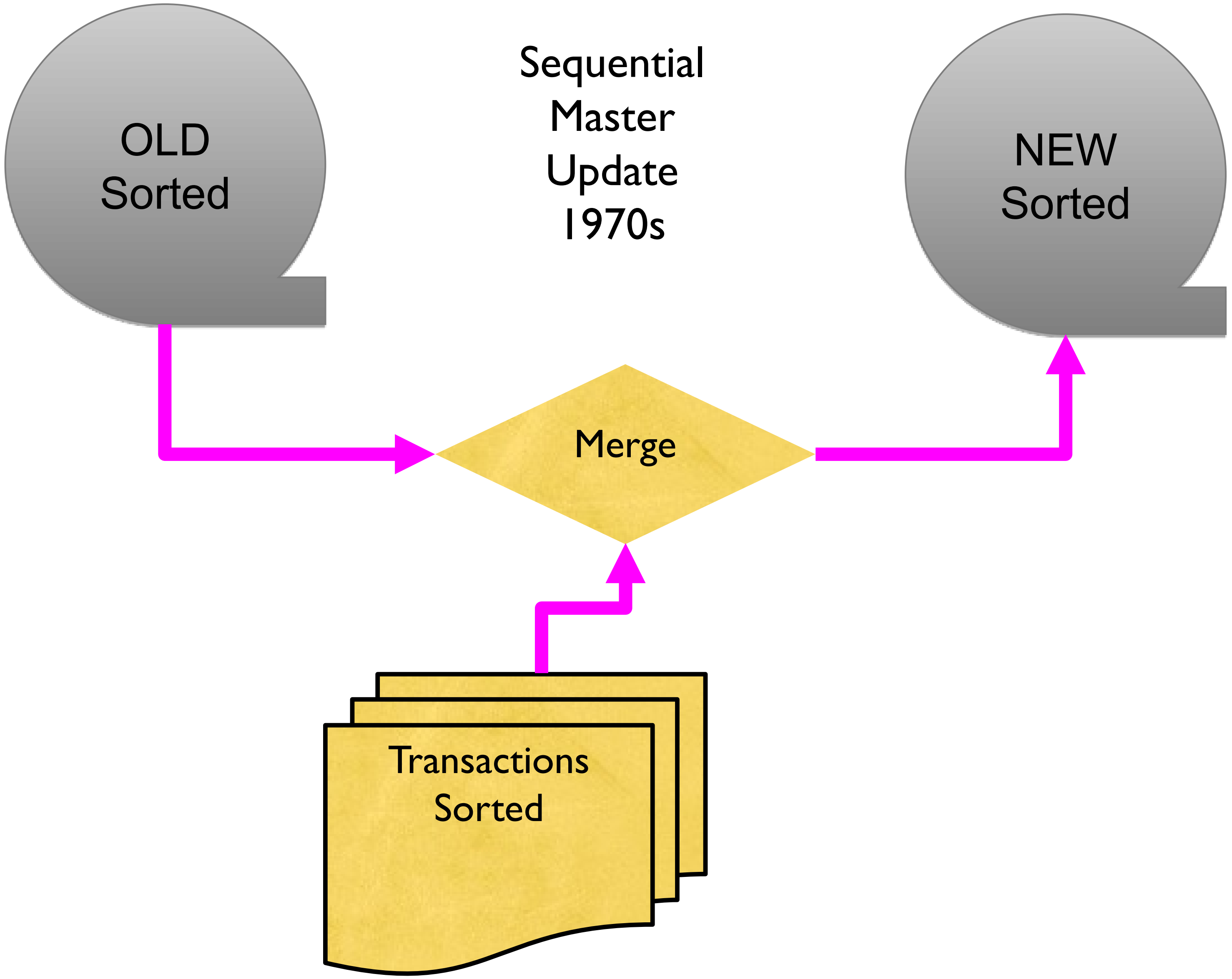
- Example

```
img = Image.open('untitled.png')
img.show()
```



DATABASES

Sequential Master Update 1970s



https://en.wikipedia.org/wiki/IBM_729

Random Access

- When you can randomly access data...
- How can you layout data to be most efficient?
- Sorting might not be the best idea

https://en.wikipedia.org/wiki/Hard_disk_drive_platter

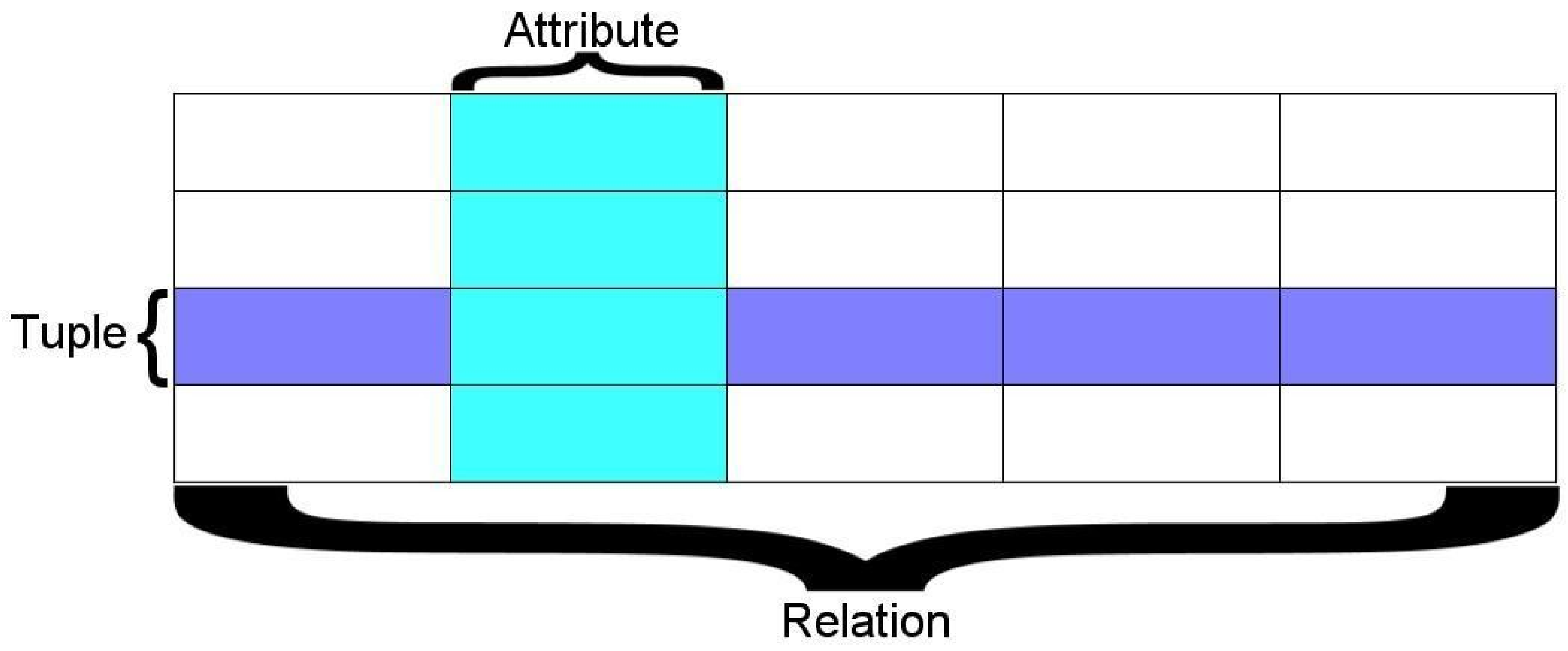
Database

- A database is a file that is organized for storing data. Most databases are organized like a dictionary in the sense that they map from keys to values.
- The biggest difference is that the database is on disk (or other permanent storage), so it persists after the program ends.
- Database software is designed to keep the inserting and accessing of data very fast, even for large amounts of data.
- Database software maintains its performance by building indexes as data is added to the database to allow the computer to jump quickly to a particular entry.
- There are many different database systems which are used for a wide variety of purposes including: Oracle, MySQL, Microsoft SQL Server, PostgreSQL, and SQLite.
- Python has an in-built support for SQLite

Relational Database

- Relational databases model data by storing rows and columns in tables.
- The power of the relational database lies in its ability to efficiently retrieve data from those tables and in particular where there are multiple tables and the relationships between those tables involved in the query.

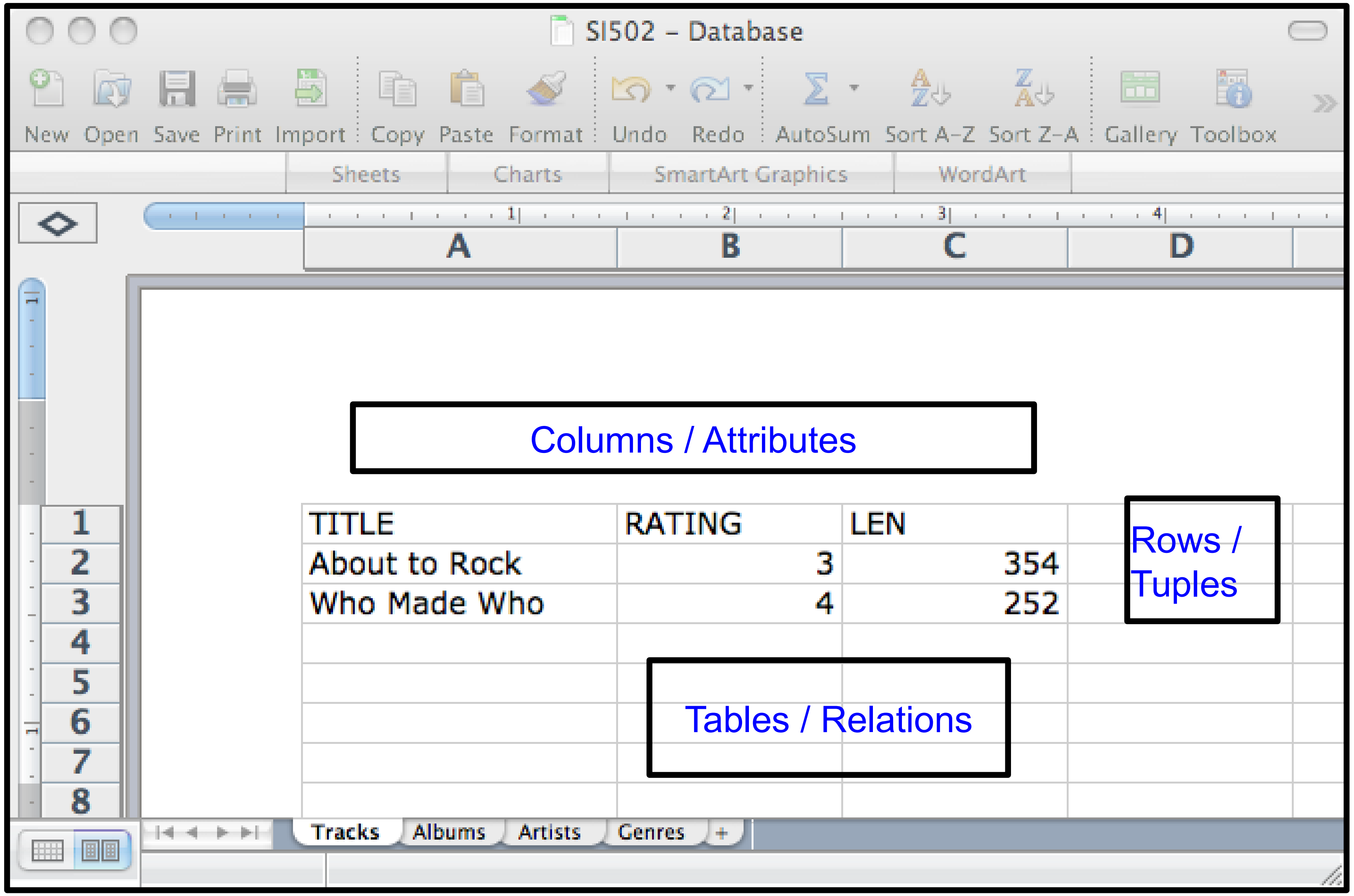
http://en.wikipedia.org/wiki/Relational_database



A relation is defined as a set of tuples that have the same attributes. A tuple usually represents an object and information about that object. Objects are typically physical objects or concepts. A relation is usually described as a table, which is organized into rows and columns. All the data referenced by an attribute are in the same domain and conform to the same constraints. (Wikipedia)

Terminology

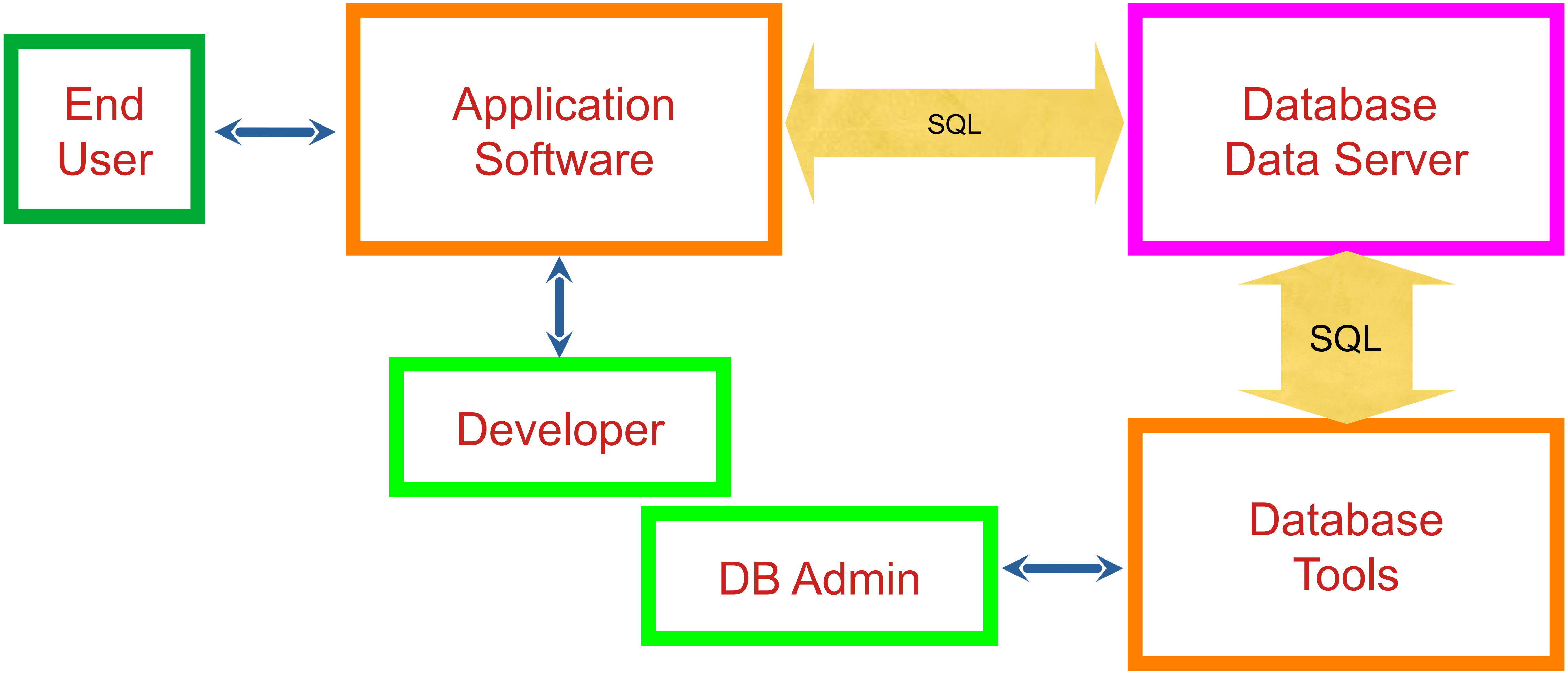
- Database - contains many tables
- Relation (or table) - contains tuples and attributes
- Tuple (or row) - a set of fields that generally represents an “object” like a person or a music track
- Attribute (also column or field) - one of possibly many elements of data corresponding to the object represented by the row



	A	B	C
1	TITLE	RATING	LEN
2	About to Rock	3	354
3	Who Made Who	4	252

Applications and Databases

- Application Developer - Builds the logic for the application, the look and feel of the application - monitors the application for problems
- Database Administrator - Monitors and adjusts the database as the program runs in production
- Often both people participate in the building of the “Data model”


End User
Application Software
SQL
Database Data Server
Developer
SQL
DB Admin
Database Tools

Database Administrator

A database administrator (DBA) is a person responsible for the design, implementation, maintenance, and repair of an organization’s database. The role includes the development and design of database strategies, monitoring and improving database performance and capacity, and planning for future expansion requirements. They may also plan, coordinate, and implement security measures to safeguard the database.

http://en.wikipedia.org/wiki/Database_administrator

Database Model

A database model or database schema is the structure or format of a database, described in a formal language supported by the database management system. In other words, a “database model” is the application of a data model when used in conjunction with a database management system.

http://en.wikipedia.org/wiki/Database_model

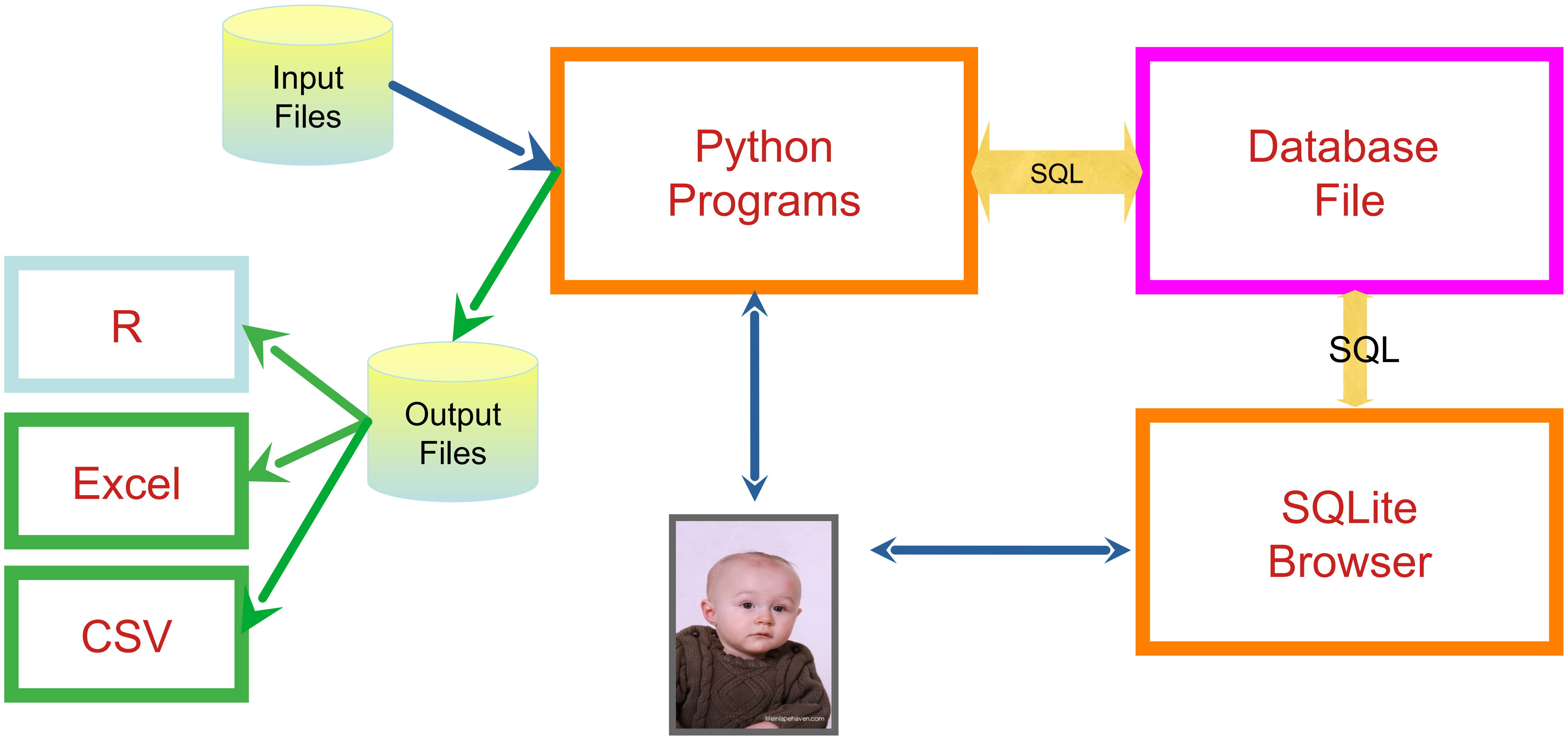
Common Database Systems

- Three major Database Management Systems in wide use
- Oracle
  - Large, commercial, enterprise-scale, very very tweakable
- MySql
  - Simpler but very fast and scalable - commercial open source
- SqlServer
  - Very nice - from Microsoft (also Access)
- Many other smaller projects, free and open source
  - HSQL, SQLite, Postgres, ...

SQL

- **S**tructured **Q**uery **L**anguage is the language
- It is used to issue commands to the database
  - **C**reate data (Insert data)
  - **R**etrieve data
  - **U**pdate data
  - **D**elete data

(Sometimes these operations are referred as ***CRUD*** operations)


Input Files
Python Programs
SQL
Database File
SQL
SQLite Browser
Output Files
R
Excel
CSV

SQLite

- SQLite is a very popular database - it is free and fast and small
- SQLite is embedded in Python and a number of other languages
- SQLite is a C library that provides a lightweight disk-based database that does not require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language.
- Some applications can use SQLite for internal data storage.
- It is also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.

SQLite is in Lots of Software...

http://www.sqlite.org/famous.html

DB Browser for SQLite – DB Management Software

- SQLite Browser allows us to directly manipulate SQLite files

  http://sqlitebrowser.org/

- DB Browser for SQLite (DB4S) is a high quality, visual, open source tool to create, design, and edit database files compatible with SQLite
- Not a visual shell for the SQLite command line tool, and does not require familiarity with SQL commands.
- Tool equally for developers and end users

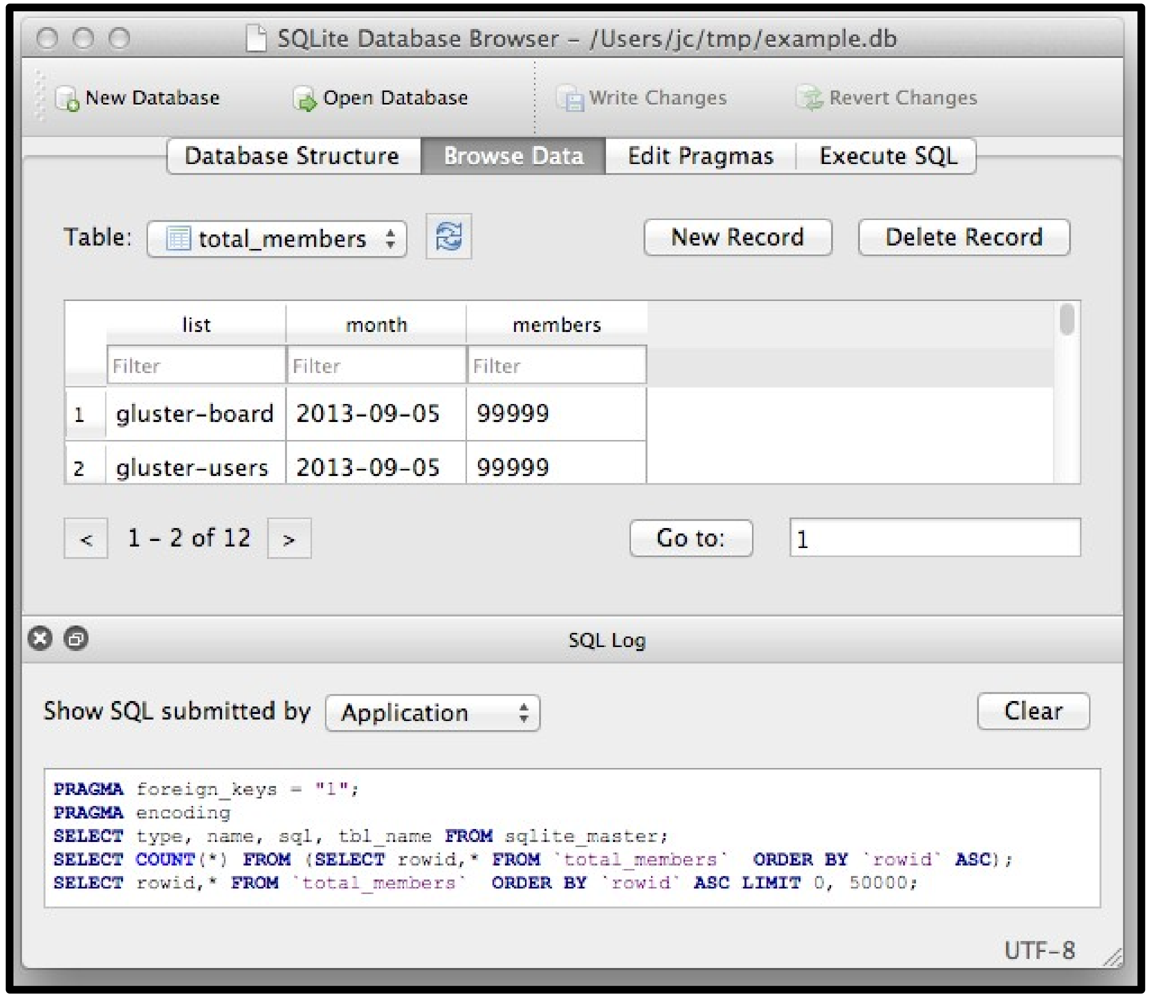


Table: total_members

	list	month	members
	Filter	Filter	Filter
1	gluster-board	2013-09-05	99999
2	gluster-users	2013-09-05	99999

< 1 - 2 of 12 > Go to: 1

Show SQL submitted by Application

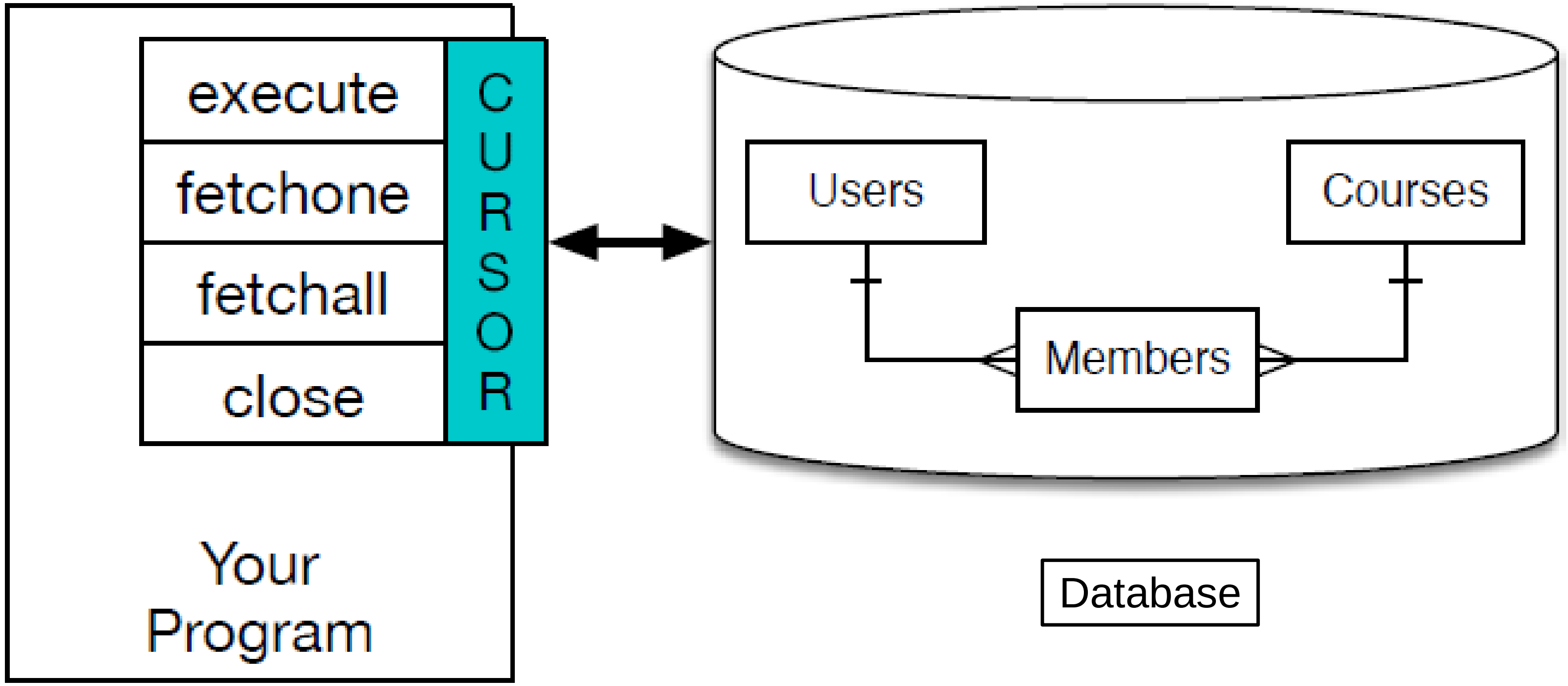
```
PRAGMA foreign_keys = "1";
PRAGMA encoding
SELECT type, name, sql, tbl_name FROM sqlite_master;
SELECT COUNT(*) FROM (SELECT rowid,* FROM `total_members`  ORDER BY `rowid` ASC);
SELECT rowid,* FROM `total_members`  ORDER BY `rowid` ASC LIMIT 0, 50000;
```

https://sqlitebrowser.org/

Working with Database

- Working with the data databases requires following steps.
  - Importing the API module.
  - Acquiring a connection with the database.
  - Issuing SQL statements and stored procedures.
  - Closing the connection

Connection and Cursor



SQLite Module in Python

- `sqlite3` is the module native to Python.
- The `sqlite3` module was written by Gerhard Häring. It provides a SQL interface compliant with the DB-API 2.0 specification described by PEP 249.
- To use the module, import it and then follow these steps;

  1) create a Connection object that represents the database
  2) create a Cursor object
  3) call `execute()` method (via Cursor object) to perform SQL commands
  4) Save (commit) the changes
  5) close the connection once done

https://docs.python.org/3.7/library/sqlite3.html

Methods

Most modules adhere to the standard - they use the same methods.

- `connect('info.db')` – to connect to the database “info”
- `cursor()` – will get a cursor object ready to execute queries.
- `execute(sql)` – an sql query
- `fetchone()` – returns just one row
- `fetchall()` – returns a list of rows
- `commit()` – saves the changes on the database.
- `rollback()` – rolls all temporary changes back.

Methods

- `close()` – closes the connection.
- `executemany()` – executes a parameterized query
- `executescript()` – executes a string of multiple SQL statements separated by a semi-colon ‘;’

SQL: Create

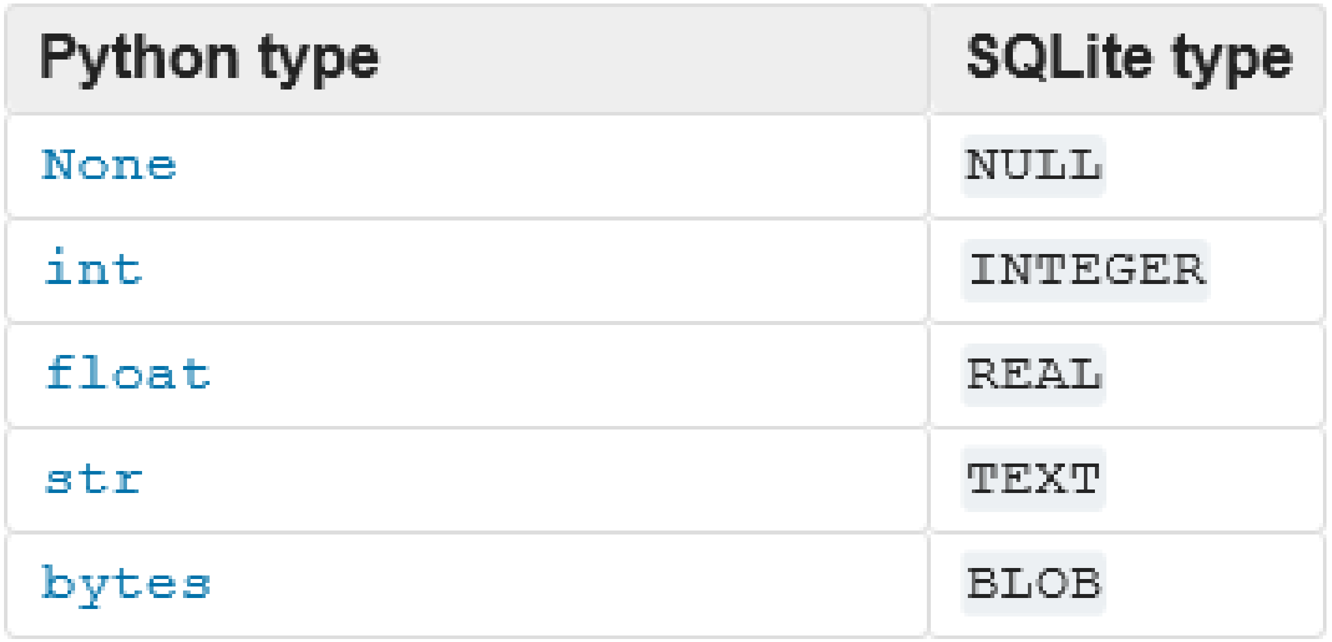
- The create statement creates new table in the database

```
CREATE TABLE Users (name TEXT, email TEXT)

CREATE TABLE Album (
   id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
   artist_id  INTEGER,
   title TEXT
   )

CREATE TABLE IF NOT EXISTS tablName(
   id INTEGER PRIMARY KEY,
   title text,
   year integer
   )
```

SQLite and Python types



Python type	SQLite type
`None`	`NULL`
`int`	`INTEGER`
`float`	`REAL`
`str`	`TEXT`
`bytes`	`BLOB`

SQL: Insert

- The Insert statement inserts a row into a table

```
INSERT INTO Users (name, email) VALUES ('Kristin', 'kf@umich.edu')
```

SQL: Update

- Allows the updating of a field with a where clause

```
UPDATE Users SET name='Charles' WHERE email='csev@umich.edu'
```

Retrieving Records

- The select statement retrieves a group of records - you can either retrieve all the records or a subset of the records with a WHERE clause

```
SELECT * FROM Users
```

```
SELECT * FROM Users WHERE email='csev@umich.edu'
```

Other logical operations allowed in a WHERE clause include <, >, <=, >=, !=, as well as AND and OR and parentheses to build the logical expressions.

SQL: Delete

- Deletes a row in a table based on selection criteria

```
DELETE FROM Users WHERE email='ted@umich.edu'
```

Example

```
import sqlite3
#create a Connection object that represents the database
conn = sqlite3.connect('example.db')
#Once you have a Connection, you can create a Cursor object and call its
execute() method to perform SQL commands
cur = conn.cursor()
data = cur.execute('SELECT name, age, email FROM users')
#Save (commit) the changes if some insert operation is performed
conn.commit()
for row in data:
   print(row)
conn.close()
```

```
('Jone', 55, 'jone@email.com')
('Chloe', 34, 'chloe@email.com')
('Stuart', 45, 'stu@gmail.com')
```

You can also supply the special name :memory: to create a database in RAM

Sorting

- You can add an ORDER BY clause to SELECT statements to get the results sorted in ascending or descending order

```
SELECT * FROM Users ORDER BY email

SELECT * FROM Users ORDER BY name DESC
```

SQLite and Python types

- Usually the SQL operations will need to use values from Python variables.
- You should not assemble your query using Python's string operations because doing so is insecure; it makes your program vulnerable to an SQL injection attack
- Instead, use the DB-API's parameter substitution.
  - Put ? as a placeholder wherever you want to use a value,
  - provide a tuple of values as the second argument to the cursor's execute() method.
  - (Other database modules may use a different placeholder, such as %s or :1.)

SQLite and Python types

```
# Never do this -- insecure!
symbol = 'RHAT'
c.execute("SELECT * FROM stocks WHERE symbol = '%s'" % symbol)

# Do this instead
t = ('RHAT',)
c.execute('SELECT * FROM stocks WHERE symbol=?', t)
print(c.fetchone())

# Larger example that inserts many records at a time
purchases = ('2006-03-28', 'BUY', 'IBM', 1000, 45.00)
c.execute('INSERT INTO stocks VALUES (?,?,?,?,?)', purchases)
```

SQL Summary

```
CREATE TABLE IF NOT EXISTS tablName(id INTEGER PRIMARY KEY, title text, year integer)

INSERT INTO Users (name, email) VALUES ('Kristin', 'kf@umich.edu')

DELETE FROM Users WHERE email='ted@umich.edu'

UPDATE Users SET name="Charles" WHERE email='csev@umich.edu'

SELECT * FROM Users

SELECT * FROM Users WHERE email='csev@umich.edu'

SELECT * FROM Users ORDER BY email

SELECT "first name" FROM students WHERE age < 25
```

THANK YOU