

CSE 503

Introduction to Computer Science for Non-Majors

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Day 27
Databases

Project Checklist

Front-End Requirements:

- ✓ HTML
- ✓ AJAX
- ✓ Callback functions

Back-End Requirements:

- ✓ Bottle routes
- ✓ Data retrieval (HTTP requests)
- ✓ Data cleaning and processing
 - Local data caching

Project Checklist


Front-End Requirements:

- ✓ HTML
- ✓ AJAX
- ✓ Callback functions

Back-End Requirements:

- ✓ Bottle routes
- ✓ Data retrieval (HTTP requests)
- ✓ Data cleaning and processing
 - **Local data caching**

We can locally cache data using text or CSV files...today we will learn how to do it with databases



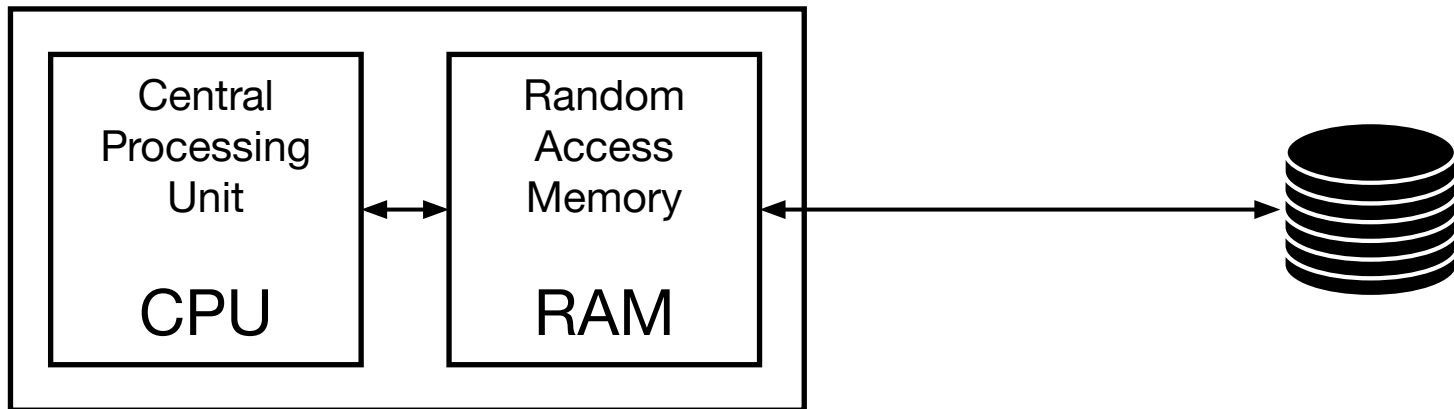
Storing Data

In Memory/CPU

- Transient (exists while program is running)
- Limited size

On Disk

- Persistent
- Larger capacity
- Text files, csv files, databases, etc



Storing Data

Text Files: Streams of characters

CSV Files: Comma separated values

Databases: Tables of data supporting highly efficient operations

(CSE 560 Data Models and Query Languages; CSE 562 Database Systems)

SQLite

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

<https://www.sqlite.org/about.html>

SQLite

SQLite is an in-process **library that implements** a self-contained, serverless, zero-configuration, transactional SQL **database engine**. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including several high-profile projects.

<https://www.sqlite.org/about.html>

SQLite - Creating and Storing a DB

```
import sqlite3

conn = sqlite3.connect('test.db')
cur = conn.cursor()

# do things to database

conn.commit()
conn.close()
```


SQLite - Creating and Storing a DB

```
import sqlite3

conn = sqlite3.connect('test.db')
cur = conn.cursor()

# do things to database

conn.commit()
conn.close()
```

1. Import the SQLite library

SQLite - Creating and Storing a DB

```
import sqlite3

conn = sqlite3.connect('test.db')
cur = conn.cursor()

# do things to database

conn.commit()
conn.close()
```

1. Import the SQLite library
2. Open a connection to a DB
(creates the DB if necessary)

Note: This file is not human readable

SQLite - Creating and Storing a DB

```
import sqlite3

conn = sqlite3.connect('test.db')
cur = conn.cursor()

# do things to database

conn.commit()
conn.close()
```

1. Import the SQLite library
2. Open a connection to a DB
(creates the DB if necessary)
3. Create a cursor object
(this is how we interact with the DB)

SQLite - Creating and Storing a DB

```
import sqlite3

conn = sqlite3.connect('test.db')
cur = conn.cursor()

# do things to database

conn.commit()
conn.close()
```

1. Import the SQLite library
2. Open a connection to a DB
(creates the DB if necessary)
3. Create a cursor object
(this is how we interact with the DB)
4. Do stuff...

SQLite - Creating and Storing a DB

```
import sqlite3

conn = sqlite3.connect('test.db')
cur = conn.cursor()

# do things to database

conn.commit()
conn.close()
```

1. Import the SQLite library
2. Open a connection to a DB
(creates the DB if necessary)
3. Create a cursor object
(this is how we interact with the DB)
4. Do stuff...
5. Commit our changes and
close the DB
(without commit, changes are lost)

So...what can we do with it?

- We can execute commands on our DB using the cursors `execute` function and passing the command we want to execute
- We will go over some basic commands today, but more details can be found on the SQLite tutorial website: <https://www.sqlitetutorial.net/>

Commands: Creating a table

Command: CREATE TABLE IF NOT EXISTS *name* *columnNames*

Commands: Creating a table

Command: CREATE TABLE IF NOT EXISTS *name* *columnNames*

name - the name of the table you want to create

columnNames - a list of names for the columns in the table

Commands: Creating a table

Command: CREATE TABLE IF NOT EXISTS *name* *columnNames*

name - the name of the table you want to create

columnNames - a list of names for the columns in the table

Example: 'CREATE TABLE IF NOT EXISTS movies (title, director, year)'

Commands: Creating a table

Command: CREATE TABLE IF NOT EXISTS *name* *columnNames*

name - the name of the table you want to create

columnNames - a list of names for the columns in the table

Example: 'CREATE TABLE IF NOT EXISTS movies (title, director, year)'

Execute with cursor (Python code):

```
cur.execute('CREATE TABLE IF NOT EXISTS movies (title, director, year)')
```

Commands: Inserting rows

Command: INSERT INTO *table* VALUES (x, y, ...z)

Commands: Inserting rows

Command: INSERT INTO *table* VALUES (*x, y, ...z*)

table - the name of the table to insert into

x, y, ...z - the values for each column

Commands: Inserting rows

Command: INSERT INTO *table* VALUES (x, y, ...z)

table - the name of the table to insert into

x, y, ...z - the values for each column

Example: 'INSERT INTO movies VALUES ("Jaws", "Spielberg", 1975)'

Commands: Inserting rows

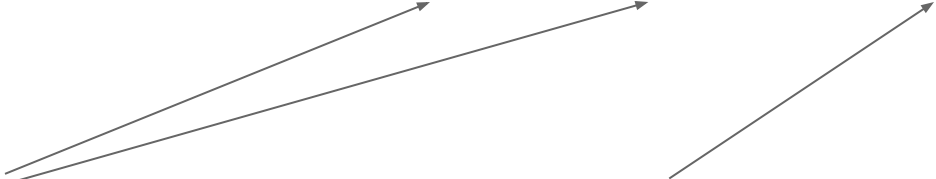
Command: `INSERT INTO table VALUES (x, y, ...z)`

table - the name of the table to insert into

x, y, ...z - the values for each column

Example: `'INSERT INTO movies VALUES ("Jaws", "Spielberg", 1975)'`

String values must be inside "", number values are just numbers
(note how we use single quotes to define the overall string)



Commands: Inserting rows

Command: `INSERT INTO table VALUES (x, y, ...z)`

table - the name of the table to insert into

x, y, ...z - the values for each column

Example: `'INSERT INTO movies VALUES ("Jaws", "Spielberg", 1975)'`

Execute with cursor (Python code):

```
cur.execute('INSERT INTO movies VALUES ("Jaws", "Spielberg", 1975)')
```

Commands: Get rows from table

Command: `SELECT * FROM table`

Commands: Get rows from table

Command: `SELECT * FROM table`

table - the name of the table to get the data from

Commands: Get rows from table

Command: `SELECT * FROM table`

table - the name of the table to get the data from

Example: `'SELECT * FROM movies'`

Commands: Get rows from table

Command: `SELECT * FROM table`

table - the name of the table to get the data from

Example: `'SELECT * FROM movies'`

Execute with cursor (Python code):

```
results = cur.execute('SELECT * FROM movies')
```

Commands: Get rows from table

Command: `SELECT * FROM table`

table - the name of the table to get the data from

Example: `'SELECT * FROM movies'`

Execute with cursor (Python code):

```
results = cur.execute('SELECT * FROM movies')
```

results is a sequence...



Commands: Get rows from table

Command: `SELECT * FROM table`

table - the name of the table to get the data from

Example: `'SELECT * FROM movies'`

Execute with cursor (Python code):

```
results = cur.execute('SELECT * FROM movies')
for entry in results:
    print(entry)
```

Commands: Get matching rows from table

Command: `SELECT * FROM table WHERE constraint`

Commands: Get matching rows from table

Command: `SELECT * FROM table WHERE constraint`

table - the name of the table to get the data from

constraint - constraint used to match specific rows

Commands: Get matching rows from table

Command: `SELECT * FROM table WHERE constraint`

table - the name of the table to get the data from

constraint - constraint used to match specific rows

Example: `'SELECT * FROM movies WHERE year = 1975'`

Commands: Get matching rows from table

Command: `SELECT * FROM table WHERE constraint`

table - the name of the table to get the data from

constraint - constraint used to match specific rows

Example: `'SELECT * FROM movies WHERE year = 1975'`

Execute with cursor (Python code):

```
results = cur.execute('SELECT * FROM movies WHERE year = 1975')
```