STATS 507
Data Analysis in Python

Week11-1: More on SQL and final project

Dr. Xian Zhang

Recap: why resurgence of DL?

1952 Stochastic gradient descent

1958 Perceptron
- Learnable weights

⋮

1986 Back propagation
- Multi-Layer perceptron

1995 Deep convolutional NN

⋮

ChatGPT… (why now)

1. Big data

- Larger dataset
- Easier collection and storage

IMAGENET

WIKIPEDIA The Free Encyclopedia

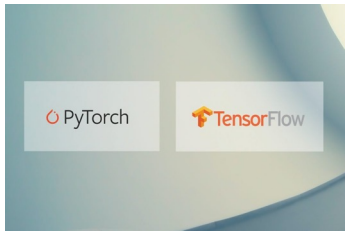
Open Dataset

2. Hardware

- GPUs
- Massively parallelizable

3. Software

- Improved techniques
- New models
- Toolboxes



Slide Credit: Alexander Amini

Modified from MIT open course: 6.S191

What is a (relational) Database?

Relational Database: a collection of tables (tabular data)

id	*name*	*address*	*class*	*dob*
12345678	Jill Jones	Canaday C-54	2011	3/10/85
25252525	Alan Turing	Lowell House F-51	2008	2/7/88
33566891	Audrey Chu	Pfoho, Moors 212	2009	10/2/86
45678900	Jose Delgado	Eliot E-21	2009	7/13/88
66666666	Count Dracula	The Dungeon	2007	11/1431
...	...	...	...	...

Each **row** in a table holds data that describes either:

- An entity
- A relationship between two or more entities

Each **column** in a table represents one **attribute** of an entity

- Each column has a domain of possible values

Writing Query with SQL?

Structured Query Language: A language via which you can create, **read**, update, and delete data in a database.

Basic extracting: `SELECT`

```
SELECT [column names] FROM [table]
```

Filter: `WHERE`

```
SELECT [column names] FROM [table] WHERE [filter]
```

Aggregating: `GROUP BY`

```
SELECT [column names],[agg functions] FROM [table] GROUP BY
[column names]
```

Ordering: `ORDER BY`

```
SELECT [columns] FROM [table] ORDER BY [column] [ASC|DESC]
```

Recap: A quick intro to SQLite

The `sqlite3` module provides an **SQL interface in Python** compliant with the database.

```
import sqlite3
con = sqlite3.connect("flights.sqlite")
```

```
cur = con.cursor()
```

Represents the connection to the on-disk database.

In order to execute SQL statements and fetch results from SQL queries, we will need to use a database cursor.

```
res = cur.execute(QUERY)
```

https://docs.python.org/3/library/sqlite3.html

More on SQLite

Final project and proposal guideline

More on `WHERE` Statements

`WHERE` supports all the natural comparisons one would want to perform

(Numerical) Operation	Symbol/keyword
Equal	=
Not equal	<>, !=
Less than	<
Less than or equal to	<=
Greater than	>
Greater than or equal to	>=
Within a range	BETWEEN … AND ...

Examples:

```
SELECT id from t_student WHERE …

        … gpa>=3.2

        … pets=1

        … gpa BETWEEN 2.9 AND 3.1

        … birth_year > 1900

        … pets <> 0
```

Caution: different implementations define `BETWEEN` differently (i.e., inclusive vs exclusive)! Be sure to double check!

More on WHERE Statements

WHERE keyword also allows set membership

Table student

id	name	gpa	major	birth_year	pets	favorite_color
101010	John Bardeen	3.1	Electrical Engineering	1908	2	Blue
314159	Albert Einstein	4.0	Physics	1879	0	Green
999999	Jerzy Neyman	3.5	Statistics	1894	1	Red
112358	Ky Fan	3.55	Mathematics	1914	2	Green

```
SELECT id, major from student WHERE major IN ("Mathematics","Statistics")

SELECT id, major from student WHERE major NOT IN ("Physics")
```

More filtering: `DISTINCT` Keyword

To remove repeats from a set of returned results:

```
SELECT DISTINCT [columns] FROM [table]
```

Table student

id	name	gpa	major	birth_year	pets	favorite_color
101010	John Bardeen	3.1	Electrical Engineering	1908	2	Blue
314159	Albert Einstein	4.0	Physics	1879	0	Green
999999	Jerzy Neyman	3.5	Statistics	1894	1	Red
112358	Ky Fan	3.55	Mathematics	1914	2	Green

```
SELECT DISTINCT pets FROM student ORDER BY pets ASC
```

Test your understanding: what should this return?

Table `student`

id	name	gpa	major	birth_year	pets	favorite_color
101010	John Bardeen	3.1	Electrical Engineering	1908	2	Blue
314159	Albert Einstein	4.0	Physics	1879	0	Green
999999	Jerzy Neyman	3.5	Statistics	1894	1	Red
112358	Ky Fan	3.55	Mathematics	1914	2	Green

```
SELECT DISTINCT pets FROM student ORDER BY pets ASC
```

pets
0
1
2

Besides basic query from 1 table…

https://www.codecademy.com/article/sql-commands

Join tables based on keys

student

ID	Name	GPA	Major	Birth Year
101010	John Bardeen	3.1	Electrical Engineering	1908
314159	Albert Einstein	4.0	Physics	1879
999999	Jerzy Neyman	3.5	Statistics	1894
112358	Ky Fan	3.55	Mathematics	1914

primary key

pets

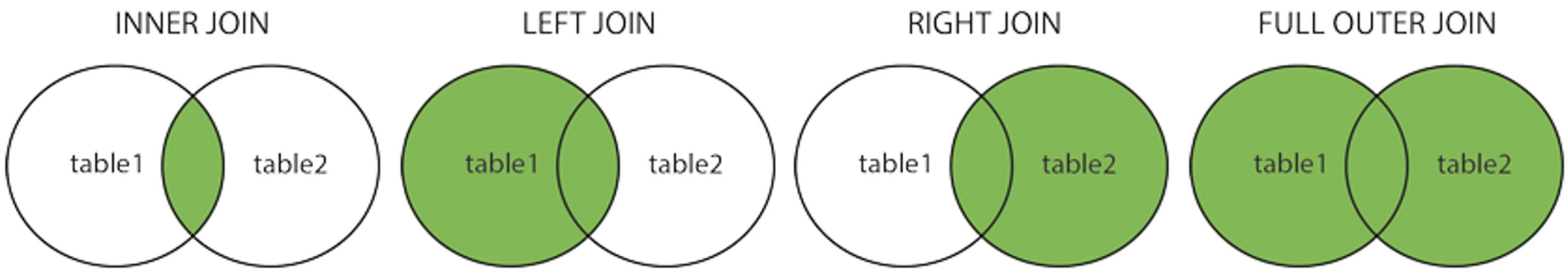
P_ID	Pet	age
101010	snoopy	3
101010	scooby	2
999999	lizzy	1
112358	loki	5

foreign key

```
SELECT id, name, pet
FROM
student INNER JOIN pets
ON id = p_id
```

ID	Name	GPA	Major	Birth Year	Pet	age
101010	John Bardeen	3.1	Electrical Engineering	1908	snoopy	3
101010	John Bardeen	3.1	Electrical Engineering	1908	scooby	2
999999	Jerzy Neyman	3.5	Statistics	1894	lizzy	1
112358	Ky Fan	3.55	Mathematics	1914	loki	5

Other ways of joining tables



`(INNER) JOIN`: Returns records that have matching values in both tables

`LEFT (OUTER) JOIN`: Return all records from the left table, and the matched records from the right table

`RIGHT (OUTER) JOIN`: Return all records from the right table, and the matched records from the left table

`FULL (OUTER) JOIN`: Return all records when there is a match in either left or right table

Image credit: https://www.w3schools.com/sql/sql_join.asp

Insert a row into a table: `INSERT INTO`

```
INSERT INTO table_name [col1, col2, col3, …]
VALUES value1, value2, value3, …
```

ID	Name	GPA	Major	Birth Year
101010	John Bardeen	3.1	Electrical Engineering	1908
314159	Albert Einstein	4.0	Physics	1879
999999	Jerzy Neyman	3.5	Statistics	1894
112358	Ky Fan	3.55	Mathematics	1914

```
INSERT INTO students (ID, Name, GPA, Major, Birth_Year)
VALUES (901010, 'Fay Ding', 3.1, Statistics’, 2000);
```

Note: if adding values for all columns, you only need to specify the values.

Modify/delete a row into a table: `UPDATE`

```
UPDATE table_name SET col1=value1,col2=value2,
WHERE condition
```

ID	Name	GPA	Major	Birth Year
101010	John Bardeen	3.1	Electrical Engineering	1908
314159	Albert Einstein	4.0	Physics	1879
999999	Jerzy Neyman	3.5	Statistics	1894
112358	Ky Fan	3.55	Mathematics	1914

Delete rows from a table:

```
DELETE FROM table_name
WHERE condition
```

```
UPDATE students SET GPA=3.8,Major= 'Physics',
WHERE Name= 'John Bardeen';
```

Caution: if `WHERE` clause is left empty, you'll delete/modify the whole table!

More on Python `sqlite3`

`Connection` object represents a database

- `Connection` object can be used to create a `Cursor` object
- `Cursor` facilitates interaction with database

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

- establish connection to given DB file (creating it if necessary)
- return `Connection` object

```
c = conn.cursor()
```

- Creates and returns a Cursor object for interacting with DB

```
c.execute( [SQL command] )
```

- runs the given command; cursor now contains query results

Python `sqlite3` in action

```python
import sqlite3
conn = sqlite3.connect('example.db')
c = conn.cursor() # create a cursor object.
c.execute('''CREATE TABLE t_student (id, name, field, birth_year)''')
students = [(101010, 'John Bardeen', 'Electrical Engineering', 1908),
            (500100, 'Eugene Wigner', 'Physics', 1902),
            (314159, 'Albert Einstein', 'Physics', 1879),
            (214518, 'Ronald Fisher', 'Statistics', 1890),
            (662607, 'Max Planck', 'Physics', 1858),
            (271828, 'Leonard Euler', 'Mathematics', 1707),
            (999999, 'Jerzy Neyman', 'Statistics', 1894),
            (112358, 'Ky Fan', 'Mathematics', 1914)]
c.executemany('INSERT INTO t_student VALUES (?,?,?,?)', students)
conn.commit() # Write the changes back to example.db
for row in c.execute('''SELECT * from t_student'''):
    print(row)
```

```
(101010, 'John Bardeen', 'Electrical Engineering', 1908)
(500100, 'Eugene Wigner', 'Physics', 1902)
(314159, 'Albert Einstein', 'Physics', 1879)
(214518, 'Ronald Fisher', 'Statistics', 1890)
(662607, 'Max Planck', 'Physics', 1858)
(271828, 'Leonard Euler', 'Mathematics', 1707)
(999999, 'Jerzy Neyman', 'Statistics', 1894)
(112358, 'Ky Fan', 'Mathematics', 1914)
```

Python `sqlite3` in action

```python
import sqlite3
conn = sqlite3.connect('example.db')
c = conn.cursor() # create a cursor object
c.execute('''CREATE TABLE t_student (id, name, field, birth_year)''')
students = [(101010, 'John Bardeen', 'Electrical Engineering', 1908),
            (500100, 'Eugene Wigner', 'Physics', 1902),
            (314159, 'Albert Einstein', 'Physics', 1879),
            (214518, 'Ronald Fisher', 'Statistics', 1890),
            (662607, 'Max Planck', 'Physics', 1858),
            (271828, 'Leonard Euler', 'Mathematics', 1707),
            (999999, 'Jerzy Neyman', 'Statistics', 1894),
            (112358, 'Ky Fan', 'Mathematics', 1914)]
c.executemany('INSERT INTO t_student VALUES (?,?,?,?)', students)
conn.commit() # Write the changes back to example.db
for row in c.execute('''SELECT * from t_student'''):
    print(row)
```

```
(101010, 'John Bardeen', 'Electrical Engineering', 1908)
(500100, 'Eugene Wigner', 'Physics', 1902)
(314159, 'Albert Einstein', 'Physics', 1879)
(214518, 'Ronald Fisher', 'Statistics', 1890)
(662607, 'Max Planck', 'Physics', 1858)
(271828, 'Leonard Euler', 'Mathematics', 1707)
(999999, 'Jerzy Neyman', 'Statistics', 1894)
(112358, 'Ky Fan', 'Mathematics', 1914)
```

Create the database file and set up a `Cursor` object for interacting with it.

Create the table. Note that we need not specify a data type for each column. SQLite is flexible about this.

Python `sqlite3` in action

```python
import sqlite3
conn = sqlite3.connect('example.db')
c = conn.cursor() # create a cursor object.
c.execute('''CREATE TABLE t_student (id, name, field, birth_year)''')
students = [(101010, 'John Bardeen', 'Electrical Engineering', 1908),
            (500100, 'Eugene Wigner', 'Physics', 1902),
            (314159, 'Albert Einstein', 'Physics', 1879),
            (214518, 'Ronald Fisher', 'Statistics', 1890),
            (662607, 'Max Planck', 'Physics', 1858),
            (271828, 'Leonard Euler', 'Mathematics', 1707),
            (999999, 'Jerzy Neyman', 'Statistics', 1894),
            (112358, 'Ky Fan', 'Mathematics', 1914)]
c.executemany('INSERT INTO t_student VALUES (?,?,?,?)', students)
conn.commit() # write the changes back to example.db
for row in c.execute('''SELECT * from t_student'''):
    print(row)
```

```
(101010, 'John Bardeen', 'Electrical Engineering', 1908)
(500100, 'Eugene Wigner', 'Physics', 1902)
(314159, 'Albert Einstein', 'Physics', 1879)
(214518, 'Ronald Fisher', 'Statistics', 1890)
(662607, 'Max Planck', 'Physics', 1858)
(271828, 'Leonard Euler', 'Mathematics', 1707)
(999999, 'Jerzy Neyman', 'Statistics', 1894)
(112358, 'Ky Fan', 'Mathematics', 1914)
```

Insert rows in the table.

Note: `sql` has special syntax for parameter substitution in strings. Using the built-in Python string substitution is insecure--vulnerable to SQL injection attack.

The `commit()` method tells `sqlite3` to write our updates to the database file. This makes our changes “permanent”

Annotated

```
1  import sqlite3
2  conn = sqlite3.connect('example.db')
```

Establishes a connection to the database stored in `example.db`.

```
3  c = conn.cursor() # create a cursor object.
```

`cursor` object is how we interact with the database. Think of it kind of like the cursor for your mouse. It points to, for example, a table, row or query results in the database.

```
4  c.execute('''CREATE TABLE t_student (id, name, field, birth_year)''')
```

`cursor.execute` will run the specified SQL command on the database.

```
13 c.executemany('INSERT INTO t_student VALUES (?,?,?,?)', students)
14 conn.commit() # Write the changes back to example.db
```

`executemany` runs a list of SQL commands.

`commit` writes changes back to the file. WIthout this, the next time you open `example.db`, the table `t_student` will be empty!

```
17 conn.close()
```

Close the connection to the database. Think of this like Python file `close`.

Metainformation: `sqlite_master`

`sqlite_master` is a special table that holds information about the "real" tables in the database

Two tables, named `t_student` and `t_thesis`

```
import os, sqlite3
os.remove('example.db') #remove old version of the database.
conn = sqlite3.connect('example.db')
c = conn.cursor()
c.execute('''CREATE TABLE t_student (id, name, field, birth_year)''')
c.execute('''CREATE TABLE t_thesis (thesis_id, phd_title phd_year)''')
for r in c.execute('''SELECT * FROM sqlite_master'''):
    print r
```

```
(u'table', u't_student', u't_student', 2, u'CREATE TABLE t_student (id, name, field, birth_year)')
(u'table', u't_thesis', u't_thesis', 3, u'CREATE TABLE t_thesis (thesis_id, phd_title phd_year)')
```

Retrieving column names in `sqlite3`

```
c.execute('''SELECT * from t_student''')
c.description
```

```
(('id', None, None, None, None, None, None),
 ('name', None, None, None, None, None, None),
 ('field', None, None, None, None, None, None),
 ('birth_year', None, None, None, None, None, None))
```

`description` attribute contains the column names; returned as a list of tuples for agreement with a different Python DB API.

```
[desc[0] for desc in c.description]
```

```
['id', 'name', 'field', 'birth_year']
```

Note: this is especially useful in when exploring a new database, like in your homework!

Final Projects (30%)

Proposal final project guidelines can be found in Files/Final Project on canvas (let's go through them together today…)

Proposal (10 pts)
Final Projects (20 pts)

Start Early!

Key dates:
Mon, 11/25/24 Proposal and preliminary results due

Wed, 12/04/24 Final project code and summary report due

Other things

HW7 is out and due next week

We will expect you learn LineCharts, BarCharts and CatPlot on your own for HW7. Read the tutorials!

Pick up your midterms

Coming next:

Intro to DP and Pytorch