SQL

Flat-file Databases

A flat file database is a *"database"* which is stored in a `.txt` or more commonly, a `.csv` file.

A way to read a `.csv` file in python you can do the following:

```python
from sys import argv


def main():
    check_args(argv)
    counts = read_csv(argv[1])
    print_languages(counts)
    exit(0)


def check_args(argv):
    if len(argv) != 2:
        print("Usage: python3 favourites.py PATH_TO_CSV")
        exit(1)


def read_csv(file_path):
    from csv import DictReader

    with open(file_path, "r") as file:
        reader = DictReader(file)
        counts = {}
        for row in reader:
            if row["language"] in counts:
                counts[row["language"]] += 1
            else:
                counts[row["language"]] = 1

    return counts


def print_languages(counts):
    for language, count in sorted(
        counts.items(), key=lambda item: item[1], reverse=True
    ):
        language = language + ((len(max(counts.keys())) - len(language)) * " ")
        print(f"{language}: {count}")


main()
```

```
# Input: python3 favourites.py ./favourites.csv
# Output:
# Python : 280
# C      : 78
# Scratch: 40
```

Relational Databases

SQL databases follow CRUD functions:

CRUD Operation	SQL Function/s
Create	CREATE, INSERT
Read	SELECT
Update	UPDATE
Delete	DELETE, DROP

To do stuff with **sqlite3** you can do the following:

- Create database from a .csv file:

```
$ sqlite3 favourites.db
SQLite version 3.45.3 2024-04-15 13:34:05
Enter ".help" for usage hints.
sqlite> .mode csv
sqlite> .import favourites.csv favourites
sqlite> .quit
$ ls -l
-rw-r--r--    1 root     root         13959 Nov  1  2023 favourites.csv
-rw-r--r--    1 root     root         24576 Jul 21 11:32 favourites.db
```

- To view the schema of a table

```
$ sqlite3 favourites.db
SQLite version 3.45.3 2024-04-15 13:34:05
Enter ".help" for usage hints.
sqlite> .schema
CREATE TABLE IF NOT EXISTS "favourites"(
"Timestamp" TEXT, "language" TEXT, "problem" TEXT);
```

- How to read data from the database

```
$ sqlite3 favourites.db
SQLite version 3.45.3 2024-04-15 13:34:05
Enter ".help" for usage hints.
sqlite> SELECT * FROM favourites;
10/30/2023 13:39:48|C|DNA
10/30/2023 13:39:50|C|Plurality
10/30/2023 13:39:53|C|Cash
10/30/2023 13:39:53|Python|Sort
10/30/2023 13:39:54|Python|DNA
10/30/2023 13:39:55|Python|Scratch
10/30/2023 13:39:56|Python|Scratch
10/30/2023 13:39:58|Python|Hello, It's Me
10/30/2023 13:39:58|C|Inheritance
```

```
10/30/2023 13:39:59|Python|Speller
10/30/2023 13:39:59|C|DNA
10/30/2023 13:40:02|Python|Readability
10/30/2023 13:40:05|Scratch|Sort
10/30/2023 13:40:06|Python|Hello, World
10/30/2023 13:40:07|Python|Filter
10/30/2023 13:40:08|Python|DNA
10/30/2023 13:40:12|C|Sort
10/30/2023 13:40:14|Python|Scrabble
10/30/2023 13:40:15|C|Sort
10/30/2023 13:40:22|C|Sort
10/30/2023 13:40:26|Python|Hello, World
```

```
$ sqlite3 favourites.db
SQLite version 3.45.3 2024-04-15 13:34:05
Enter ".help" for usage hints.
sqlite> SELECT language FROM favourites;
Python
Python
Python
C
Python
C
Python
Scratch
Python
Python
Python
C
Python
C
C
Python
Python
Python
C
C
Python
```

- To truncate the results:

```
$sqlite3 favourites.db
SQLite version 3.45.3 2024-04-15 13:34:05
Enter ".help" for usage hints.
sqlite> SELECT language FROM favourites LIMIT 10;
Python
Python
```

```
Python
Scratch
Python
Python
Python
Python
Python
Python
```

Some keywords in sqlite include:

Keyword	Definition	Example
`AVG`	Get the average	
`COUNT`	Count how many things there are	`SELECT COUNT(*) FROM favourites;` count all rows in `favourites` table. `SELECT COUNT(DISTINCT(language)) FROM favourites;` will return the amount of unique languages
`DISTINCT`	Get unique values	`SELECT DISTINCT(language) FROM favourites;` will return a table of all unique languages
`LOWER`	Force everything to lowercase	`SELECT DISTINCT(LOWER(language)) FROM favourites;` will return all the unique languages in lowercase
`UPPER`	Force everything to uppercase	`SELECT DISTINCT(UPPER(language)) FROM favourites;` will return all the unique languages in uppercase
`MAX`	Get maximum value	
`MIN`	Get minimum value	
`WHERE`	Simmilar to a conditional to filter results	`SELECT COUNT(*) FROM favourites WHERE language = 'C';` will return the count of rows where `language` is 'C'. To get count of people who liked 'C' as well as 'Hello, World' you can do `SELECT COUNT(*) FROM favourites WHERE language = 'C' AND problem = 'Hello, World';`
`LIKE`	Allows you to use wildcards to filter results	`SELECT problems, COUNT(*) AS n FROM favourites WHERE problems LIKE 'C%' GROUP BY problems ORDER BY n DESC;` will give you the count of problems that begin with 'C'

Keyword	Definition	Example
`ORDER BY`	Orders the table by a table heading (default is ASC)	`SELECT language, COUNT(*) FROM favourites GROUP BY language ORDER BY COUNT(*) DESC;` will output a table of favourite languages and their counts from most to least favourite
`LIMIT`	Limits the amount of rows in the output	`SELECT language, COUNT(*) AS n FROM favourites GROUP BY languages ORDER BY n DESC LIMIT 1;` will give you the most popular language and it's count
`GROUP BY`	Create a table where it is grouped by something	`SELECT language, COUNT(*) FROM favourites GROUP BY language;` will give you a table of language to the count of that language
`AS`	A way to alias items to not repeat yourself	`SELECT language, COUNT(*) AS n FROM favourites GROUP BY language ORDER BY n DESC;`

- To insert into a table you can do:

```
INSERT INTO table (column, ...) VALUES(value, ...);
```

For example, to add a new entry to the table:

```
INSERT INTO favourites (language, project) VALUES('SQL', 'Fiftyville');
```

The Timestamp field will be *"NULL"*

- To delete from a table you can do:

```
DELETE FROM table WHERE condition;
```

For example, to delete the field we just made:

```
DELETE FROM favourites WHERE Timestamp = 'NULL';
```

- To update exisiting entries in the table you can use:

```
UPDATE table SET column = value WHERE condition;
```

For example, to update entries where their favourite language was *C* to have their favourite language be *SQL* and their favourite topic to be *Fiftyville*, you can do:

```
UPDATE favourites SET language = 'SQL', problem = 'Fiftyville' WHERE language = 'C';
```

IMDB

You can use `JOIN` to join two relational tables together. For example, given two tables `shows` and `ratings` where `ratings` has a `show_id` field. You can get shows of ratings above 6.0 by doing the following:

```
SELECT title, rating FROM shows JOIN ratings ON shows.id = ratings.show_id WHERE rating >= 6

-- OUTPUT from shows.db:
-- +-----------------------------+--------+
-- |            title            | rating |
-- +-----------------------------+--------+
-- | Zeg 'ns Aaa                 | 6.7    |
-- | Catweazle                   | 7.8    |
-- | UFO                         | 7.9    |
-- | Ace of Wands                | 7.7    |
-- | The Adventures of Don Quick | 7.7    |
-- | All My Children             | 6.8    |
-- | Archie's Funhouse           | 6.8    |
-- | Arnie                       | 7.1    |
-- | Barefoot in the Park        | 7.2    |
-- | The Best of Everything      | 8.1    |
-- +-----------------------------+--------+
```

Indexes

To time queries, you can use `.timer` in `sqlite3`

You can index a column of a table to be able to perform queries faster as you have prepared it in advance. For example:

```sql
CREATE INDEX title_index ON shows (title);
```

This will create an index of the title's column in the shows table. You will need to update and maintain this column over time.

`id` columns are indexed automatically, however foreign keys are not. Therefore, to find all the shows that *Steve Carell* is in, we need to index the `stars.show_id`, `stars.person_id`, and `name` columns. This can be done and searched by the following:

```sql
-- Only needs to be completed once
CREATE INDEX person_index ON stars (person_id);
CREATE INDEX shows_index ON stars (show_id);
CREATE INDEX name_index ON people (name);

-- Query for Steve Carell
SELECT title FROM shows, stars, people
WHERE shows.id = stars.show_id
AND people.id = stars.person_id
AND name = 'Steve Carell';

-- +-------------------------------------+
-- |                title                |
-- +-------------------------------------+
-- | The Dana Carvey Show                |
-- | Over the Top                        |
-- | Watching Ellie                      |
-- | Come to Papa                        |
-- | The Office                          |
-- | Entertainers with Byron Allen       |
-- | The Naked Trucker and T-Bones Show  |
-- | Some Good News                      |
-- | ES.TV HD                            |
-- | Inside Comedy                       |
-- | Rove LA                             |
-- | Metacafe Unfiltered                 |
-- | Fabrice Fabrice Interviews          |
-- | The Office: Superfan Episodes       |
-- | Riot                                |
-- | Séries express                      |
-- | Hollywood Sessions                  |
-- | First Impressions with Dana Carvey |
-- | LA Times: The Envelope             |
-- | Space Force                        |
-- +------------------------------------+
-- Run Time: real 0.000 user 0.000217 sys 0.000117
```

The issue with this is that you have to use so much more space for the data structure.

Python and SQL

So, to use SQL in Python, you need to use the `sql` package.

Race Conditions

A race condition is when a variable is being updated on some state and then another user is trying to update it while it is in the process of being updated. For example, if you get home and find you have no milk, you go out and get some milk, then, before you get home, a roomate finds that there is no milk so they go out to get milk. Now you have twice as much milk as you needed, it's going to go sour and, well, no use crying over spilled milk. The solution is to use some SQL syntax:

`BEGIN TRANSACTION` and `COMMIT` can be wrapped around a SQL query to avoid these issues.

SQL injection attacks

SQL injection is when a user uses an input that can be translated into a different SQL query. For example, for a login, if the query is:

```
import sqlite3

db = sqlite3.connect("users.db").cursor()

username = input("Username: ")
password = input("Password: ")

res = db.execute(f"SELECT user, pass FROM usernames WHERE user = '{username}' AND pass = '{p
if res.fetchone() is not None:
    unlock()
```

This will work well if someone gives `John` and `Password` if it is in the database, however, if someone was to give `John'--` and `x` it will comment the part of the SQL query that is checking for the password, and this will be able to get through. This is why you should always use the built-in placeholders as they will correctly sanitise your queries for you:

```
import sqlite3

db = sqlite3.connect("users.db").cursor()

username = input("Username: ")
password = input("Password: ")

res = db.execute("SELECT user, pass FROM usernames WHERE user = ? AND pass = ?", username, p
if res.fetchone() is not None:
    unlock()
```