Module 0

Exercise 1: Your own key-value store

1. Create a bash script named `mykv.sh`. This script accepts three parameters, `make`, `set` and `get`:
    1. `./mykv.sh make [table]` creates a `table` (actually an empty file in the current directory with the name supplied)
        - 💡 you could use `touch`
    2. `./mykv.sh set [table] [key] [value]` inserts a record with the supplied `key` and `value` and stores it on a line in the `table` file
        - 💡 you should use `>>`
    3. `./mykv.sh get [table] [key]` returns the current record with the supplied `key` in the `table`. If multiple records exist with the same key, it should only return the **last one**. If none exist, it should return `NULL`
        - 💡 you should use `grep`, `sed` and `tail`
2. Insert some records in a new table and check the functionality
    - What is the **worst-case** complexity of each operation?
    - Explain.
3. Implement a new endpoint: `./mykv.sh del [table] [key]` that deletes the record with the supplied `key` in the `table` so it will appear as `NULL` the next time you `get` it.
    - What could be at least two different approaches to implement this?
        - 💡do you **really** need to **actually** delete the data?
    - What would be their performance characteristics?
4. Implement a `./mykv.sh get [table] cleanup` that performs the actual deletion of data that is **marked** as deleted
5. Benchmark your implementation in terms of **average operations per second** for get, set and del
    - 💡you can use commands such as `openssl rand -hex 16` to generate random strings
6. Add a `sync -d` call after every write (using `&&`) to the file in the `set` code path to ensure **durability**
    - Read the `man sync`. Why does this help ensure durability?
    - Re-run your benchmarks. What do you see?
    - What do you think you should do to mitigate this?
7. **Advanced** you may now want to ensure that only one process can write to a table at the same time. Read about the `flock` syscall/utility and how you could use it guarantee this. Implement this in your routines.

Exercise 2: Are you still good at SQL?

1. Given those two simple tables:

Animal
Dog
Lion

Animal
Elephant

Animal
Cat
Tiger
Dog

Which query would give the following output?

Animal	Animal
Dog	Dog
Lion	NULL
Elpehant	NULL
NULL	Cat
NULL	Tiger

2. Let's define an `ExamResults` table

Student	Ewam	Pass?
Steve	Maths	true
Steve	Physics	true
Steve	English	false
Steve	CS	true
Eleanor	CS	false

Knowing that to pass in next year, students have to pass all the exams in the `MandatoryExams` table:

Exam
Maths
Physics
CS

How can you output the list of distinct students, with a boolean indicated that they passed their year or not? (There are **many** solutions here)

3. A query to debug

The following query is producing "wrong results": users complain that some clients do not show up. Moreover, the average_order_amount value is different that in another query that uses the `AVG()`

function. FInally "cost" metric is also completely **wrong**. How would you fix it?

```
SELECT
  customer_name
  SUM(o.amount) / COUNT(0) AS average_order_amount,
  SUM(o.shipping_cost + o.cost) AS cost
FROM
  customers c
  LEFT JOIN orders o
  ON o.customer_id = c.customer_id
WHERE
  o.amount < 10000
  AND
  c.country IN ('France', 'UK', NULL)
GROUP BY
  customer_name
```