Felipe Lima - Cadence

PART 1: Web Development & Automation

1. **You are responsible for developing a Linux Machines Monitor using PHP and MySQL. You will need a main table called Machine where every row represents a real Linux machine and contains detailed information about it.**

    a. **How would you construct the DB model? Draft an ER model. Include all attributes you judge necessary.**

    In order to construct a DB model for the specified Linux Machines Monitor, the main table "Machine" (where every row represents a real Linux machine) would contain the following format:

    - Machine (MachineID (PK), Hostname, IP, OS, RAM, CPU, HDD, Graphics, Display, Hardware, Uptime, Status)

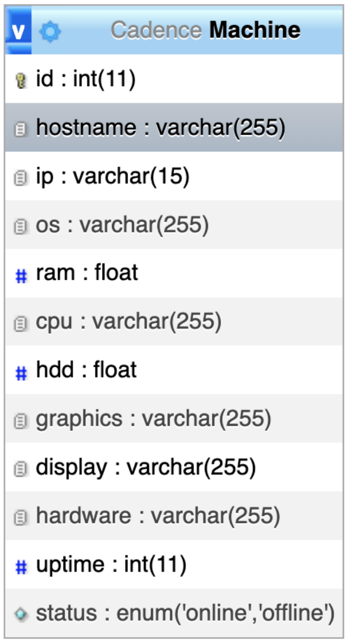
    Attributes:

    - `id`: This is the primary key for the table and is used to uniquely identify each machine. The serial number can be used or a new ID can be created.
    - `hostname`: The hostname of the Linux machine.
    - `ip`: The IP address of the Linux machine.
    - `os`: The version of the Linux operating system running on the machine.
    - `ram`: The amount of RAM installed on the machine.
    - `cpu`: The type and number of CPUs installed on the machine.
    - `hdd`: The amount of storage space installed on the machine.
    - `graphics`: The type of graphics installed on the machine.
    - `display`: The display installed on the machine.

- `hardware`: Type of hardware the Linux OS is running on.
- `uptime`: The uptime of the machine (in seconds)
- `status`: The current status of the machine (e.g. online, offline, etc.).

The following code represents the creation of the database `cadence` and the creation of the table `Machine`

```
-- Create schema for exercise
DROP SCHEMA IF EXISTS `Cadence`; CREATE SCHEMA IF NOT EXIS
TS `Cadence`; USE `Cadence` ;
-- Create a table called Machine with the
-- following columns:
DROP TABLE IF EXISTS Machine;
CREATE TABLE IF NOT EXISTS Machine (
      id INT AUTO_INCREMENT PRIMARY KEY,
      hostname VARCHAR(255) NOT NULL,
      ip VARCHAR(15) NOT NULL,
      os VARCHAR(255) NOT NULL,
      ram FLOAT NOT NULL,
      cpu VARCHAR(255) NOT NULL,
      hdd FLOAT NOT NULL,
      graphics VARCHAR(255) NOT NULL,
      display VARCHAR(255) NOT NULL,
      hardware VARCHAR(255) NOT NULL,
      uptime INT NOT NULL,
      status ENUM('online', 'offline') NOT NULL
  );
```



Cadence **Machine**
id : int(11)
hostname : varchar(255)
ip : varchar(15)
os : varchar(255)
ram : float
cpu : varchar(255)
hdd : float
graphics : varchar(255)
display : varchar(255)
hardware : varchar(255)
uptime : int(11)
status : enum('online','offline')

ER model for 'Cadence' DB

b) A Web page will be necessary to handle the Machines information, using a data table and web forms. Write a detailed description of how you would insert machines on the database, query it to retrieve the needed information, instantiate the objects from the table and fill the HTML table considering a page filter form. Include all the code used to build the page using pure PHP for the backend and HTML, CSS, JavaScript or React for the frontend.

To create said Web page I would:

1. Create an HTML file containing :

    a. A form for user input on new Machines.
    b. A filter form
    c. A table view
2. Create a series of php scripts to handle:
    a. Creation of a table to store the Linux machines informations (create_table.php)
        i. Using the sql query mentioned above, I created a table through a php connection to the database.
    b. Addition of new machines to the table (add_machine.php)
        i. Through JavaScript the information is collected from the user on the web page and then directed through a POST request to a php file that uses the information to create a query and executes it.
    c. Reseting table (reset_table.php)
        i. Simple php script that executes a query that empties the current Machine table.
    d. Getting the DB information from the server and display it on the web page
        i. This is done by executing a simple `SELECT * FROM Machine` query and returning the result to JS as a JSON object so it can be used to populate the table.
    e. Filtering the table to display only desired entries according to specifications
        i. Script receives information from web page through a GET request and applies it to a query that filters the entries on the table and returns the results as a JSON object.
3. Create a JavaScript file that is responsible for:
    a. Event listeners such as button clicks.
    b. Collecting information from the web page through user input.
    c. Redirecting this information to the php scripts.
    d. Populating the results in the web page table.

4. Create a server to be hosted locally for testing as well as a connection to a MySQL database. This is done through XAMPP, which handles all the connections.

The step by step the program follows is:

1. Creates a connection to a MySQL database and to an Apache Web Server through XAMPP.
2. On the web page the user create a table by clicking on the `Create Machine Table` button
    a. This sends an on click signal to a JS function `createTable()` that sends a POST request to `create_table.php` which in turn executes the query that creates the `Machine` table on the already set up `Cadence` database.
3. The user is prompted to insert the details of the machine to be added on a series of form entries and once all of them are filled (fields have a NOT NULL property on the DB), clicking on the `Add Machine` button sends a signal to a JS function `addMachine()`. The function retrieves the information from the web page and through a POST requests and sends the information to `add_machine.php` which queries it and executes the query to add the machine to the DB.
    a. On success the function clear the form entries and makes a call to `getMachines()`.
4. `getMachines()` then gets the DB information from the server and display it on the web page
    a. This is done by executing a simple `SELECT * FROM Machine` query and returning the result to JS as a JSON object so it can be used to populate the table.
    b. On success the function clears the table body and then repopulates it with the machines in the DB.
5. If the user decides to filter the information on the table, the filter form works similarly to the `getMachines()` function. It collects data from the web page and sends a GET request to `filter_machines.php`.
    a. Script receives information from web page through a GET request and applies it to a query that filters the entries on the table and returns the results as a JSON object.

💡 NOTE: All the code is on GitHub for review.

c) Another responsibility is the creation of a process to monitor the Linux machines
periodically. You need to check if all machines are up, but only some of them need to have
their usage monitored. The monitoring consists of:

1. **Get the list of machines that need testing**
2. **Check if each machine is up**
3. **Check the logged users**
4. **Check the running processes**
5. **Check CPU and memory usage**

Explain in detail how you would retrieve the machine list, how you would execute each of
those tests (commands, flows, etc.) and how you would store the results in the database
(refer to the DB model asked on question 1a). Do you think there is any other relevant
information to monitor?

To execute the described tests I would follow these steps:

1. Retrieve the list of machines that need testing
    a. To do so, I would query all the machines that are set to test. I could use a query like:
    `SELECT * FROM Machines WHERE monitor = true`
2. After this information is retrieved and placed into an array, I would iterate through it for each:

3. Check if each machine is up
    a. Using the retrieved ip address from the machines that need monitoring, I would use the `exec` and `ping` command to find what machines are up and reachable. Next store the return information.
4. Check the logged users
    a. To check the logged users I would use the `exec` and `who` command which would return the logged users and store those.
5. Check the running processes
    a. To check for the running processes on the machine the `ps aux` command can be used and once again store the results.
6. Check CPU and memory usage
    a. CPU and Memory usage can be retrieved through the `top` command.
7. Store the results in the database
    a. Create a new table with the status of the machines such as `machine_status` by using a create table query like the one mentioned above.
        i. Each column on the new table represents an information requested (machine_id, status, logged_users, running_processes, usage).
    b. Use an INSERT statement on a query to add the desired information into the new table, using all the stored data from the collection above.

Some other relevant information to monitor include:

- Network traffic
- Energy consumption
- Disk usage

2) You are responsible for a critical system and needs to update a software package with minimum downtime and impact. How would you proceed with the upgrade? Explain the migration plan, covering any possible challenges.

A series of steps must be taken in order to guarantee a smooth, fast, secure and successful upgrade. The first thing I would make sure to have is a back up of the system. This would allow me to prepare for any eventuality and in the case of an issue rolling out the update, I would have a rollback plan to restore a previous working version of the system.

To minimize downtime and the chance of encountering issues, I would plan and test the upgrade before fully deploying it. Testing it in a controlled environment can be useful to identify and solve issues that my arise. Planning involves schedule the upgrade during low usage times, which minimizes the impact on the users and makes it easier to fix any complications on the process.

Making use of a rollout upgrade, a canary or blue-green deployment can also be beneficial in reducing downtime and impact. These would allow for a gradual upgrade by either running two identical systems in parallel or upgrading a small subset of the system first and then gradually the rest of the system.

PART 2: Python

3) A certain system needs a password validator module, which upon receiving a string with a
password and a list with the requirements of this password, returns whether the password is valid or not. The list of the password requirements is composed of tuples containing the following:

- **First value:**
  - **LEN – password length**
  - **LETTERS – # of letters**
  - **NUMBERS – # of numbers**
  - **SPECIALS – # of special characters**
- **Second value: <, > or =**
- **Third value: an integer number**

  Ex.:

req = [('LEN', '=', 8), ('SPECIALS', '>', 1)]
req specify a password with eight characters and at least two special characters.

Write a Python 3 script to solve this problem and the unit test to validate it, without installing
external packages.

```
# Name: Felipe Lima
# Assessment for Cadence Design Systems 01/11/2023

# Find the number of characters in the password that match the specified character type
def find_len(password, char_type):
    if char_type == "LEN":
        return len(password)
    elif char_type == "LETTERS":
        return len([char for char in password if char.isalpha()])
    elif char_type == "NUMBERS":
        return len([char for char in password if char.isdigit()])
    elif char_type == "SPECIALS":
        return len([char for char in password if not (char.isalpha() or char.isnumeric
())])


#Validates the password agaisnt the given parameters
def validator(password, reqs):
    for req in reqs:
        char_type, operator, value = req
        if (operator == "="):
            if(find_len(password, char_type) != value):
                return False
        if (operator == "<"):
            if(find_len(password, char_type) >= value):
                return False
        if (operator == ">"):
            if(find_len(password, char_type) <= value):
                return False
    return True

# Prints whether the password id valid or not
def is_valid(password, reqs):
    if validator(password, reqs) == True:
        print("Password is valid")
    else:
        print("Password is not valid")


# Testing the password validator
def test_validator():
    # Test for password length
    reqs = [('LEN', '=', 8)]
    assert validator("password", reqs) == True
    assert validator("p@ssword", reqs) == True
    assert validator("dkslp02h", reqs) == True
    assert validator("12e#$89s", reqs) == True
    assert validator("djkfn43fjk4!", reqs) == False
    assert validator("", reqs) == False
    assert validator("!", reqs) == False

    # Test for # of letters
    reqs = [('LETTERS', '>', 5), ('LETTERS', '<', 15)]
    assert validator("password", reqs) == True
    assert validator("p@ssword", reqs) == True
    assert validator("12345678", reqs) == False
    assert validator("asdf404", reqs) == False
    assert validator("djkfnddddjjdjdjdjdjdjdjdjdjdj43fjk4!", reqs) == False
    assert validator("", reqs) == False
    assert validator("!", reqs) == False

    # Test for # of numbers
    reqs = [('NUMBERS', '>', 5), ('NUMBERS', '<', 15)]
    assert validator("password", reqs) == False
    assert validator("p@ssword", reqs) == False
    assert validator("12345678", reqs) == True
    assert validator("2349asdf404", reqs) == True
    assert validator("sjd0101940fj((!", reqs) == True
    assert validator("12345678901234567890", reqs) == False
    assert validator("!", reqs) == False

    # Test for # of specials
    reqs = [('SPECIALS', '>', 5), ('SPECIALS', '<', 15)]
    assert validator("password", reqs) == False
    assert validator("p@ssword", reqs) == False
    assert validator("12@#$%^!345678", reqs) == True
    assert validator("!!**djdjdjd&*@", reqs) == True
    assert validator("!*!*!", reqs) == False
    assert validator("ahdj******@@@(@(@(@(@(@(@(@@@@@", reqs) == False
    assert validator("!", reqs) == False

    # Test for all
    reqs = [('LEN', '>', 8), ('LETTERS', '>', 4), ('NUMBERS', '=', 3), ('SPECIALS', '=',
 1), ]
    assert validator("password", reqs) == False
    assert validator("p@ssword", reqs) == False
    assert validator("passw123!", reqs) == True
    assert validator("pawi!234", reqs) == False
    assert validator("!apelo234", reqs) == True
    assert validator("!apeljdjdfjdjjdfjnefeo234", reqs) == True
    assert validator("", reqs) == False
    assert validator("!", reqs) == False

test_validator()
```

💡 NOTE: The python code is also on GitHub

PART 3: Quality Assurance

4) Imagine that you are responsible for guaranteeing the quality of a software that is constantly updated. How would you guarantee that those updates will not affect parts of the software that were working correctly before?

Here are some of the steps I would take in oder to guarantee the quality of a software that is constantly updated, and to make sure the updates will not affect parts of the software that were working correctly before:

- Version control: I would make use of some sort of version control such as git, to make sure I can always go back to previous working versions of the code, and that my new changes don't conflict with the existing code.
- Testing:
    - Writing unit tests can be really useful in ensuring the new code follows the desired instructions and outputs the correct information.
    - Regression testing: previously run test cases are re-executed to verify that the functionality is still effective. This guarantees that any new code modifications have no negative consequences on the functionality already in place. It guarantees that when the most recent code changes are made, the older code still functions.
- Making use of CI tools like Travis CI, that provides fast feedback on the success of the modification while supporting your development process by automatically creating and testing code changes.