DQuaT - Database Query Automation Tool

A final report submitted for the course named Project III(CS-400)

BY

PRANAV BIRENDRA PATHAK

Bachelor of Technology, VII Semester

Roll No. 17010112

Under the Supervision and Guidance of

Dr. NAVANATH SAHARIA

Department of Computer Science and Engineering

Indian Institute of Information Technology Senapati, Manipur

February, 2021

Abstract

The aim of this project is to build Query Automation software tool to create database infrastructure. The main focus area through this project is here users need to define the query of databases in key-value configuration language yml. In real world, data is not stored in one single database as one database is not sufficient to solve the data problems for that we have tons of different database with different use case. But the problem is each database has their own query language to operate and it is difficult for one to learn every language, sometime query gets long and confusing. This one tool smart enough to interact multiple databases at one time and can setup the entire database with one click. Since the formate is in key-pair, no confusion and easy to write all database query in single go.

Keywords - Database, Python, YAML

Declaration

I declare that this submission represents my idea in my own words and where others' idea or words have been included, I have adequately cited and referenced the original source. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/sources in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from proper permission has not been taken when needed.

(Signature)
(Pranav Birendra Pathak)
Date: (17010112)

Department of Computer Science & Engineering
Indian Institute of Information Technology Manipur

Certificate

This is to certify that the project report entitled **"DQuaT - Database Query Automation Tool"** submitted to the Department of Computer Science and Engineering, Indian Institute of Information Technology, Manipur, in partial fulfillment for the award of the degree of **Bachelor of Technology** in Computer Science and Engineering, is a record of bona fide work carried out by **Mr. Pranav Birendra Pathak**, Roll No. 17010112.

No part of this report has been submitted elsewhere for award of any other degree.

(Dr. Navanath Saharia)

Assistant Professor
Supervisor

Date:

Department of Computer Science & Engineering
Indian Institute of Information Technology Manipur

Certificate

This is to certify that the project report entitled **"DQuaT - Database Query Automation Tool"** submitted to the Department of Computer Science and Engineering, Indian Institute of Information Technology, Manipur, in partial fulfillment for the award of the degree of **Bachelor of Technology** in Computer Science and Engineering, is a record of bona fide work carried out by **Mr. Pranav Birendra Pathak**, Roll No. 17010112.

No part of this report has been submitted elsewhere for award of any other degree.

Dr. Nongmeikapam Kishorjit Singh

Assistant Professor and Head,

Department of CSE

IIIT, Manipur

Date:

Department of Computer Science & Engineering
Indian Institute of Information Technology Manipur

Certificate

This is to certify that the project report entitled **"DQuaT - Database Query Automation Tool"** submitted to the Department of Computer Science and Engineering, Indian Institute of Information Technology, Manipur, in partial fulfillment for the award of the degree of **Bachelor of Technology** in Computer Science and Engineering, is a record of bona fide work carried out by **Mr. Pranav Birendra Pathak**, Roll No. 17010112.

Signature of HoD
(Dr. Nongmeikapam Kishorjit Singh)

Signature of Examiner 1: _____

Signature of Examiner 2: _____

Signature of Examiner 3: _____

Acknowledgement

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. I would like to extend my sincere thanks to all of them. I am highly indebted to **Dr. Navanath Saharia** for his guidance and constant supervision on my project **"DQuaT - Database Query Automation Tool"** as well as for providing necessary information regarding the project and also for their support in completing the project. I would like to express my gratitude towards my parents for their kind co-operation and encouragement which help me in completion of this project. I would like to express my special gratitude and thanks to my mentor for giving me such attention and time. My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

- Mr. Pranav Birendra Pathak

Contents

Abstract ii

Declaration iii

Certificate iv

Certificate v

Certificate vi

Acknowledgement vii

Table of contents viii

List of figures xi

1 Introduction 2

1.1 File System 3

1.2 Database Management System 4

1.3 Need of this project 4

1.4 Objective 5

1.4.1 Gantt chart 5

2 Existing System Study 7

2.1 Existing System. 8
2.2 Problems In Existing system 9
2.3 Observations 10
2.4 Summary 10

3 System Analysis, Design & Implementation 11

3.1 Introduction 12
3.2 System Designing 12
3.3 Methodology and Implementation 14
3.3.1 Hosts 14
3.3.2 Playbook 16
3.3.3 Engine 17
3.3.4 How to write a code in YML 17
3.3.5 How to write Playbook in DQuaT 19
3.3.6 How to write Host File in DQuaT 20
3.3.7 Available Modules in DQuaT 20
3.4 Summary 21

4 Conclusion 22

4.1 Overall Benifits 23

Appendix A Screenshot and Description of the Modules to be Implemented 24

A.1 Create 24
A.2 Insert 25
A.3 Grant 26
A.4 Drop 27

A.5 Alter 28
A.6 Command 29
A.7 Delete 30
A.8 Lock 31
A.9 Rename 31
A.10 Revoke 32

Appendix B User manual **33**

B.1 Introduction 33
B.2 Steps for setup 33

List of Figures

1.1 Project Timeline 6

3.1 System working Design Architecture Overview 13

3.2 System interaction with different databases 13

3.3 Host file format 15

3.4 Host file format with Multiple groups 15

3.5 Playbook for multi-hosts multi-database 16

3.6 YML file for basic syntax 18

3.7 YML dick-key 18

3.8 Complex dick-key 19

3.9 Boolean in yml 19

3.10 Basic format for Playbook 20

A.1 CREATE 24

A.2 INSERT 25

A.3 These are the images with code for grant module 26

A.4 These are the images with code for drop module 27

A.5 These are som more images with code for drop module 28

A.6 Alter 29

A.7 command 29

A.8 delete 30

A.9 Lock 31
A.10 Rename 31
A.11 Revoke 32

Chapter 1

Introduction

“Let Each and Every cell of your body fulfill your desire”

- Pranav Birendra Pathak

1.1 File System

By data, we mean known facts and statistics collected together for reference or analysis and that have implicit meaning. The earliest system used to store these data or groups of records in separate files, and so they were called **File systems**. File system organizes the files and helps in retrieval of files when they are required. File systems consists of different files which are grouped into directories. Typically, each department has its own folders and files, designed specifically for those applications. The department itself working with the data processing staff, sets policies or standards for the format and maintenance of its files.

Some of Advantages of using File System to store data

1. Companies mainly use file processing to handle large volumes of structured data on a regular basis.

2. File processing design approach was well suited to mainframe hardware and batch input.

3. File Processing can be more efficient and cost less than a DBMS in certain situation.

4. The design of file processing is more simple than design of Database.

5. File processing cost less and can be more speed than Database.

6. We can customize file processing more easily and efficiently than Database because files are related with the application and it have all the data needed for that application.

With the increased importance of data and security, these old approach were not usefull. Its has many disadvantages. **Some of very important disadvantages are:**

1. Security problems

2. Integrity problems

3. Atomicity of updates

4. Difficulty in accessing data

5. Data isolation - multiple files and formats

6. Concurrent access by multiple users

7. Data redundancy and inconsistency

To overcome all the above problem, here comes a tool that manages the collection of related data called **Database Management System.**

1.2 Database Management System

A database can be defined as a collection of coherent, meaningful data. It is used for storing data and retrieving the data effectively when it is needed. DBMS is a software for managing the database. In Database Management System the data can be fetched by **SQL queries** and **relational algebra.**

Benefits of DBMS

1. Reduced data redundancy

2. No more data inconsistencies

3. Stored data can be shared by a single or multiple users

4. Data integrity can be maintained

5. Security of data can be simply implemented

6. Standards can be set and followed

7. Data independence can be achieved

1.3 Need of this project

In a real world scenario, one single database cannot solve the all the problems related to data or big data. Big firms or MNCs uses different set of database to solve their data problems. Example: Facebook uses Mysql, HBase, Cassandara, Haystack, Memcache for cache database and many more. Similarly, if we talk about google, google itself use tons

of different database for different use case, eg: Mysql, Bigtable, Spanner, Google cloud, Firestore and many more.

To Manage data, we have tons of database avaiable with different use case, mainly categorised as:

1. **SQL**: Mysql, Pg/SQL, Postgres, MariaDB and manymore
2. **NoSQL**: MongoDb, Cassandra, CouchDB, Hypertable, Redis, Riak, Neo4j, Hadoop HBASE and many more

Each database has its own query language to retrieve or store or basically manipulate the data. Each query has its own syntax and way of writing and executing query. It is harder for an individual to study all the query or to remember all the different syntax of different databases.

1.4 Objective

Keeping the above focus on mind, DQuaT is a solution. It is a query automation software tool to create database infrastructure. It will automate the internal database query. Here users need to define the query of databases in key-value configuration language similar to yml. This major focus on this project will be given on:

1. Interact multiple databases in one go.

2. Simplicity in query writing.

1.4.1 Gantt chart

The Figure 1.1 shows the steps taken to implement the system.The box represent that the work has been done untill the end of that block.The Whole system is divided into 5 phases some of them i have done in a parallel fashion which can be seen in the graph.

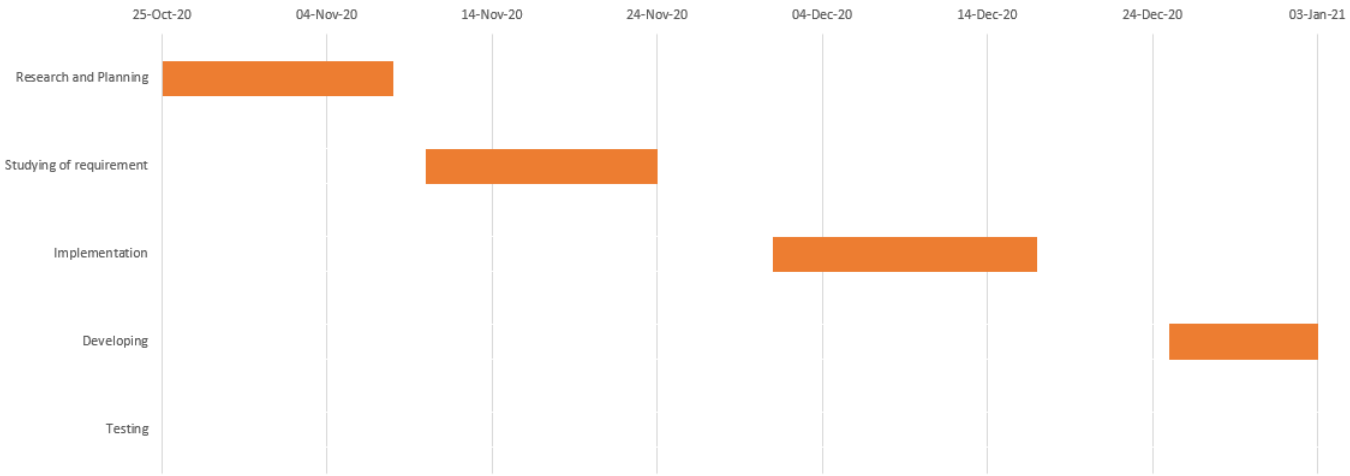

Figure 1.1: Project Timeline

Chapter 2

Existing System Study

Outline: This chapter presents the following:

1. A brief introduction about existing system.
2. Problems in existing system.

2.1 Existing System.

Everything that comes to device are **data**. Internet is full of data. Every business runs on data, Without it there cannot be business on internet, whether its a small mini application or big data application sources such as Media, Cloud, IOT, Web. **Bussines** of Banks, General Electric, Social media company like Facebook, Instagram, Twitter, Medium, LinkedIn.....many many more never-ending list are totally depends on Data Manipulation.

If you do little more research on data

1. 4 petabytes of data are created on Facebook per day

2. 4 terabytes of data are created from each connected car per day

3. 500 million tweets are sent per day

4. 294 billion emails are sent per day

5. 65 billion messages are sent on WhatsApp.

6. Google gets over 3.5 billion searches per day ie If you break this statistic down, it means that Google processes over 40,000 search query every second on average.

7. 1.7MB of data is created every second by every person during 2020.

8. Google is working on is the self-driving car. Using and generating massive amounts of data from sensors, cameras, tracking devices, and coupling this with on-board and realtime data analysis from Google Maps, Streetview and other sources allows the Google car to safely drive on the roads without any input from a human driver.

9. Huge amounts of data are recorded from every aircraft and every aspect of ground operations, which is reported in real-time and targeted specifically to recovering from disruption and returning to a regular schedule

But to store these data whether structured or unstructured, we use **Database**. It's important to know when to use which database according to types of data,

1. When data is structured and efficient to process and store, we use **SQL-Structured Query Language** .

2. When Data that includes unstructured and semi-structured, also called **BIG DATA**, we use **NoSQL** databases.

2.2 Problems In Existing system

Data on internet is so big and also fall on different categories of data, it is not impossible for a single database to process all the data. so in real world, we do not store data in one single database as one database is not sufficient to solve the data problems. For this purposes we have tons of different database available with different use case.

The problem with using different database are

1. They have their own query language to operate

2. Complexity increases with the length of query and finds difficult to troubleshoot.

3. Problem due to Documentation.

There is no doubts that we should use different databases for different application as per their use case. Since every database that their own query language, it is difficult for one person to learn all the query languages for different databases. The problems do not end here, Query of databases gets more complicated as its length increases. When using databases in real world, Most of the query length exceed length 2, which lead to complexity in query over a period of time and slightly errors can lead to headache for troubleshooting.

Query we execute on databases, or set of query used to create a database infrastructure, has been documented for future or for reference. If due to some technical issue, system got crash or there is a need for same database infrastructure on other machine, we need to again create the database infrastructure by looking the document. while doing so, there is very less probability that they can succeed in first attempt. One survey shows that 60 out of 100 require min 5 attempt to successfully execute the document. Some of the problems faced by process of document over code are:

1. Expensive as it is time consuming

2. Sometimes hard to read and understand

3. Sometimes incomplete and not updated

2.3 Observations

I have observed that my project that is **DQuaT - Database Query Automation Tool** would be of immense help to the user as it is fully CLI, solve all the problems listed above and give a new, better and fast approach to automate the entire database infrastructure with one click. This project will be an ideal solution for database user.

2.4 Summary

In this chapter, we describe the existing system for deployment of database infrastructure and the problems faced by it. One of the Key Factors in this fast running era is speed and automation. You must be always be ready for scale in out your production. Here comes the role of this project. With one click entire database infrastructure will created and can be deleted.

Chapter 3

System Analysis, Design & Implementation

Outline: This chapter presents the following:

1. A brief of system design.
2. A brief of methodology.

3.1 Introduction

In this ever growing technology and advancement world, automation is everywhere. Everything needs to be automated with one click with minimum user interference in one click. DQuaT has a potential to automate the entire database infrastructure. This one tool smart enough to interact multiple databases at multiple location at one time and can setup the entire database with one click. Since the format is in key-pair, no confusion and easy to write all database query in single go.

.

3.2 System Designing

This DQuaT is a database automation tools where user just need to tell where is database located , name of databases and tasks what to perform on that database. By giving these information, with one click entire database infrastructure an be created.

The system is developed using the Python language. Module used here till now are:

1. **pyyml**: This module is used to convert the yml file into python list and dictionary
2. **pymongo**: This module is used to connect between Client and Mongodb databases
3. **pymysql**: This module is used to connect between Client and MySql databases
4. **psycopg2**: This module is used to connect between Client and mysql databases

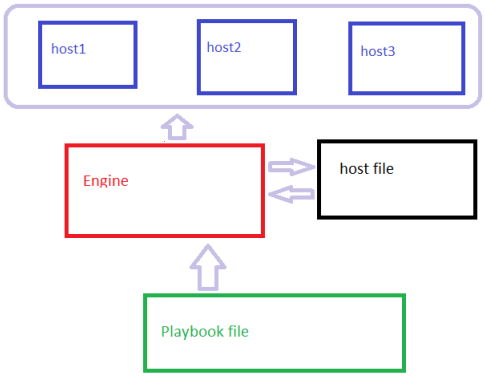

Figure 3.1: System working Design Architecture Overview

Figure 3.2 shows the working architecture of this tools. It is divided into 3 layer. Every time DQuaT is executed, it goes through these layer.

1. **Layer 1:** This layer contains Playbook file. All the playbook file to be run are in this layer.

2. **Layer 2:** This layer contains engine and hosts file. Engine will retrieve the data from playbook. First engine will retrieve the groups of hosts mentioned in playbook. According to the group name, engine will search that group in host file and retrieve the all the hosts details present in that particular group.

3. **Layer 3:** This layer contains physical hosts or database. As per the hosts details retrieved from host file at layer 2, engine will start to execute the tasks in series at respective hosts.

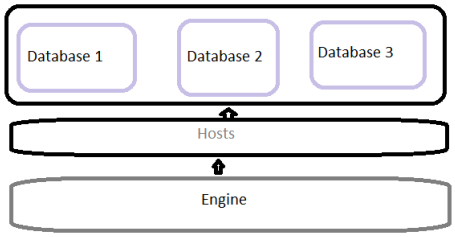

Figure 3.2: System interaction with different databases

Figure 3.2 shows the working architecture of engine connecting with different databases. Engine first refer to host file to get about the location of databases. After getting the location, engine will run the tasks according to the type of database mentioned on playbook.

3.3 Methodology and Implementation

With the aim to build a smart tool that interact multiple databases at multiple location, DQuaT has 3 main components.

1. **Hosts (Inventory)**
2. **Playbook**
3. **Engine**

3.3.1 Hosts

DQuat works against multiple Databases or "hosts" in your infrastructure at the same time, using a list or group of lists known as inventory. Once your inventory is defined, you use patterns to select the hosts or groups you want DQuat to run against.

The default location for inventory is a file name "hosts.yml" in your current dirctory. You can also use multiple inventory files at the same time, and/or pull inventory from dynamic or cloud sources but it should be in YAML.

Inventory basics: formats, hosts, and groups

The inventory file should be in YAML formats. A basic YAML in hosts.yml might look like this:

```
- group: mix
  member1:
      host: 13.233.136.145
      user: jvie
      passwd: "#jview@123"
      database: jdb

  member2:
      host: 192.168.43.88
      user: testuser
      passwd: test123!
      database: test4db
```

Figure 3.3: Host file format

Figure 3.3 shows host file basic format. Here group is a fixed keyword whereas member of groups can be of any name. For every member host, user, passwd, and database are the fixed keyword.

One host file can have multiple groups shown in figure 3.4

```
- group: local
  host1:
      host: 192.168.43.88
      user: testuser
      passwd: test123!
      database: test4

- group: mix
  member1:
      host: 13.233.136.145
      user: jvie
      passwd: "#jview@123"
      database: jdb
```

Figure 3.4: Host file format with Multiple groups

3.3.2 Playbook

DQuaT Playbooks offer a repeatable, re-usable, simple database management and multi-machine deployment system, one that is well suited to deploying complex database. If you need to execute a task with DQuaT more than once, write a playbook and put it under source control.

Playbook syntax and execution

Playbooks are expressed in YAML format with a minimum of syntax. A playbook is composed of one or more 'tasks' in an ordered list. Each task calls an DQuaT module and executed on respective hosts.

```
- group: remote
  dbName: mysql
  tasks:
  - name: create database 1
    create:
        type: database
        name: test4

  - name: create table
    create:
        type: table
        name: ttt5
        columns:
            - firstname VARCHAR(30) NOT NULL
            - lastname VARCHAR(30) NOT NULL
            - email VARCHAR(50)

- group: remote
  dbName: mongodb
  tasks:
  - name: create database 2
    create:
        type: database
        name: test2
  - name: create table 2
    create:
        type: table
        name: ttt21
        columns:
            - firstname VARCHAR(30) NOT NULL
            - lastname VARCHAR(30) NOT NULL
```

Figure 3.5: Playbook for multi-hosts multi-database

A playbook runs in order from top to bottom. Within each play, tasks also run in order from top to bottom. Playbooks with multiple 'plays' can orchestrate multi-machine deployments, running one play on your database1, then another play on your

database2, then a third play on your database 3, and so on. Figure 3.5 shows playbook for multi-hosts and multi-database.

By default, DQuaT executes each task in order, one at a time, against all machines matched by the host pattern. Each task executes a module with specific arguments. When a task has executed on all target machines, DQuaT moves on to the next task

To run your playbook, use the command: "quat playbookfile.yml"

3.3.3 Engine

DQuaT engine works by connecting to your database hosts and pushing out tasks, called "modules" to them through respective database modules. It is the soul of the product among the 3. Engine takes 2 input that is playbook file and hosts file,and gives output on the databases by executing the tasks.

Engine read playbook file in YML format through the module "pyyml", convert that file into python data type called list and dict and then proceed with further work to execute tasks as per group of hosts mentioned by going to hosts file, which was second input to engine. Engine with respect to the hosts of group, keep the list of connection to that host with the details mentioned in subsection of host file.

3.3.4 How to write a code in YML

Before writing the code, lets first know about the working syntax of YAML which is basic overview of correct YAML syntax, this is how Engine interpret playbooks (our DQuaT language) are expressed.

Why YAML?

We use YAML because it is easier for humans to read and write than other common data formats like XML or JSON.

YAML Syntax

For DQuaT, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a “hash” or a “dictionary”. So, we need to know how to write lists and dictionaries in YAML.

There’s another small quirk to YAML. All YAML files (regardless of their association with DQuaT or not) can optionally begin with — and end with .... This is part of the YAML format and indicates the start and end of a document.

All members of a list are lines beginning at the same indentation level starting with a ”- ” (a dash and a space) Figure 3.6

```
---
# A list of name
- mack
- jack
- whak
- black
...
```

Figure 3.6: YML file for basic syntax

A dictionary is represented in a simple key: value form (the colon must be followed by a space) Figure 3.7

```
# An employee record
martin:
  name: Martin D'vloper
  job: Developer
  skill: Elite
```

Figure 3.7: YML dick-key

More complicated data structures are possible, such as lists of dictionaries, dictionaries whose values are lists or a mix of both Figure 3.8

```yaml
# Employee records
- martin:
    name: Martin D'vloper
    job: Developer
    skills:
      - python
      - perl
      - pascal
- tabitha:
    name: Tabitha Bitumen
    job: Developer
    skills:
      - lisp
      - fortran
      - erlang
```

Figure 3.8: Complex dick-key

We will also gonna use bolean while writng the code, which will be done as Figure 3.9

```yaml
create_key: yes
needs_agent: no
knows_oop: True
likes_emacs: TRUE
uses_cvs: false
```

Figure 3.9: Boolean in yml

3.3.5 How to write Playbook in DQuaT

There are 4 reserved Keywords to know before proceeding to write playbook:

1. **group:** Every file starts with group. It indicate the name of group of host where to execute
2. **dbName:** It says name of database for the given hosts

3. **name:** This key is not gonna processed, it is used for user reference. This will get printed as per its position
4. **tasks:** This is where actual tasks lies. Every tasks start with the name of module Figure 3.10 shows how these keywords should be noted

```
- group: remote
  dbName: mysql
  tasks:
  - name: create database 1
    create:
```

Figure 3.10: Basic format for Playbook

3.3.6 How to write Host File in DQuaT

Writing host file is very simple. It has only 1 reserved keyword to remember, ie **group**.

Host file start with group keywords, it denote the name of group. It is the same name that mentioned in playbook. Every group has host. Group will continue with list of hosts. There can be any number of hosts inside one group shown in Figure 3.4

Every hosts has 4 keywords, which is required to do basic connection to database.

1. **hosts:** It is the host name of host where database is running
2. **user:** User name responds to name of user inside database
3. **passwd:** It responds to password of that user needed to authenticate.
4. **database:** It is the name of default database to work upon.

3.3.7 Available Modules in DQuaT

This tool is intelligent due to its module present. Each module do specific works on database. Every tasks calls the specific modules used to do the respective tasks.

List of the module present inside **MySQL** are **create, insert, grant, alter, select, drop, truncate, revoke, lock, command, rename, update, delete and select**

while **create** and **insert** are the module present for database **PgSql and Mongodb**

3.4 Summary

DQuaT is a database infrastructure tools wich can automate the entire infrastructure in one click as this tool smart enough to interact multiple databases at one time at multple location and can setup the entire database with one click. For the simplicity of the user, it is made in a key-value pair which solves the complexity of query in real world. Currently, module present is of MySql, pgsql, and mongodb.

Chapter 4

Conclusion

After understanding the wholes sole purpose of database and big data, the best way to manage database resources or database infrastructure for now is to use DQuaT. DQuaT is a cli tools that can very well suited for **CI/CD** (Continous Integration/ Continous Deployment) for **DevOps** approach. There is no doubt that manually creating is a good way to manage and use the database but it is also an older approach as well as slower, more complex code and difficult to refer. This tool is an optimal solution to all the above mentioned problems.

4.1 Overall Benifits

With single language, with single file, with simpler approach we can automate the entire database infrastructure.

1. **Code over Document**: We make document of every task, but while executing through document, it is harder to achieve result in one go. With one go, one click we can achieve the result through code. Since code is written by developer itself, this helps in covering all the small details and all the possible case, scenario. It will error free and efficient to use.
2. **Simple approach**: It maybe confusing sometimes to write big query, but if the format is like key-value pair, there will be no confusion everything will be clear cut. Complexity canbe solved easily using this approach.
3. **Single language**: One do not need to learn all query language. if they know the logic and overview, this yml way is enough. Learning this one language which is easier to any language till known.

Appendix A

Screenshot and Description of the Modules to be Implemented

A.1 Create

```
- group: remote
  dbName: mysql
  tasks:
  - name: create database 1
    create:
        type: database
        name: test4

  - name: create table
    create:
        type: table
        name: ttt5
        columns:
            - firstname VARCHAR(30) NOT NULL
            - lastname VARCHAR(30) NOT NULL
            - email VARCHAR(50)
  - name: create role
    create:
        type: role
        role:
             - 'admin'
             - 'student'
             - "'automation'@'localhost'"
```

Figure A.1: CREATE

Figure A.1 is the defined code for the create module.

A.2 Insert

```
- group: aws
  dbName: mysql
  tasks:
  - name: insert into table 1
    insert:
        table: ttt2
        columns:
            - firstname
            - lastname
        values:
            - "'pranav', 'pathak'"
            - "'abc', 'efg'"

  - name: insert into table 2
    insert:
        table: ttt1
        values:
            - "'a' , 'b' , 'a@b'"
            - "'c' , 'd' , 'c@d'"
            - "'e' , 'f' , 'e@f'"
```

Figure A.2: INSERT

Figure A.2 is the defined code for the insert module

A.3 Grant

Figure A.3 is the defined code for the grant module playbook.

```
- group: aws
  dbName: mysql
  tasks:
  - name: grant 1
    grant:
        type: role
        roles:
            - 'admin'
            - 'student'
        user_or_role:
            - gg

- group: aws
  dbName: mysql
  tasks:
  - name: grant 2
    grant:
        type: privileges
        privileges:
            - select
            - create
        database: "*"
        table: "*"
        user_or_role:
            - gg
```

```
- group: aws
  dbName: mysql
  tasks:
  - name: grant 3
    grant:
        type: proxy
        proxy_user_or_role:
            - name
        grant_option: no

        user_or_role:
            - gg
```

Figure A.3: These are the images with code for grant module

A.4 Drop

Figure A.4 A.5 is the defined code for the drop module playbook.

```
- group: remote
  dbName: mysql
  tasks:
  - name: drop from mysql
    drop:
        type: table
        name:
            - "newName"
        temporary: yes
        restrict: yes
        cascade: no
  - name: drop from mysql
    drop:
        type: database
        name:
            - "newName"
  - name: drop from mysql
    drop:
        type: schema
        name:
            - "newName"
```

```
  - name: drop from mysql
    drop:
        type: function
        name:
            - "newName"
  - name: drop from mysql
    drop:
        type: procedure
        name:
            - "newName"
  - name: drop from mysql
    drop:
        type: index
        index_name:
            - "newName"
        table_name:
            - "newName"
        algorithm: default
        lock: default
```

Figure A.4: These are the images with code for drop module

```yaml
- name: drop from mysql
  drop:
      type: logfile
      logfile_group:
          - "newName"
      engine: "engine_name"

- name: drop from mysql
  drop:
      type: server
      server_name:
          - "name"

- name: drop from mysql
  drop:
      type: spatial_reference_system
      srid:
          - 4210

 - name: drop from mysql
   drop:
       type: tablespace
       tablespace_name:
           - "name"
       undo: yes
       engine: "engine_name"

 - name: drop from mysql
   drop:
       type: trigger
       schema_name: "name"
       trigger_name:
           - "name"
 - name: drop from mysql
   drop:
       type: view
       view_name:
           - "name"
       restrict: yes
       cascade: yes
 - name: drop from mysql
   drop:
       type: event
       event_name:
           - "newName"
```

Figure A.5: These are som more images with code for drop module

A.5 Alter

Figure A.6 is the defined code for the alter module

```yaml
- group: remote
  dbName: mysql
  tasks:
  - name: alter table ttt1
    alter:
        table: ttt1
        add:
            - id varchar(255)
            - middle varchar(255)
            - DateOfBirth year
  - name: delete lastname
    alter:
        table: ttt1
        drop:
            - lastname
  - name: modify table column
    alter:
        table: ttt1
        modify:
            - DateOfBirth date
  - name: rename column
    alter:
        table: ttt1
        renameColumn:
            - email emailer VARCHAR(100)  NULL
  - name: rename table name
    alter:
        table: ttt1
        renameTable:
            - newName
```

Figure A.6: Alter

A.6 Command

```yaml
- group: remote
  dbName: mysql
  tasks:
  - name: executing query directly
    command:
        query:
            - "show databases"
            - "show tables"
        commit: no
```

Figure A.7: command

Figure A.7 is the defined code for the command module

A.7 Delete

```
- group: aws
  dbName: mysql
  tasks:
  - name: delete table
    delete:
        low_priority: yes
        ignore: yes
        quick: yes

        partition_name_list:
            - partition1
            - partition2
        table_reference:
            - ttt2
        from_table_reference:
            - t1 INNER JOIN t2 INNER JOIN t3
        using_table_reference:
            - t1 INNER JOIN t2 INNER JOIN t3
        where_condition:
            - condition
        order_by: id

        limit_row_count: 1
```

Figure A.8: delete

Figure A.8 is the defined code for the delete module

A.8 Lock

```
- group: remote
  dbName: mysql
  tasks:
  - name: lock table
    lock:
        database: test1
        table_name:
            - tt1
            - tt1 as t1
        lock_type:
            - read
            - write
```

Figure A.9: Lock

Figure A.9 is the defined code for the lock module

A.9 Rename

```
- group: aws
  dbName: mysql
  tasks:
  - name: rename table
    rename:
        old_table_name:
            - ttt1
        new_table_name:
            - newttt1
```

Figure A.10: Rename

Figure A.11 is the defined code for the rename module

A.10 Revoke

```yaml
- group: aws
  dbName: mysql
  tasks:
  - name: alter table ttt1
    revoke:
        type: role
        roles:
             - 'admin'
             - 'student'
        user_or_role:
            - gg


- group: aws
  dbName: mysql
  tasks:
  - name: alter table ttt1
    revoke:
        type: privileges
        privileges:
            - select
            - create
        database: "*"
        table: "*"
        user_or_role:
            - gg
```

Figure A.11: Revoke

Figure A.11 is the defined code for the revoke module

Appendix B

User manual

B.1 Introduction

This application is fully CLI based. User have to write the code in YAML format. Since the format is in key-value pair, it will be user friendly. Since it is made entirely on python, it will be platform or OS independence

Setting up the environment is very easy and straight forward since there is no external changes has to made on database side. User just need python library for respective database.

B.2 Steps for setup

1. Run the Setup.py file on python with the command **python setup3.py**. It will automatically do all the necessary requirement.

2.Or manually put the files to /bin folder and make it executable by the command **chmod +x quat.py**

3. Your setup is done. Start executing the playbook with the command
quat playbook.yml

Bibliography

1. Bernardo, Susan; Murphy, Graham J (2006). Ursula K. Le Guin : a critical companion. Westport, Conn: Greenwood Press. p. 18. ISBN 9780313027307. OCLC 230345464.
2. Chatham, Mark (2012). Structured Query Language By Example - Volume I: Data Query Language. p. 8. ISBN 978-1-29119951-2.

Bibliography

1. Bernardo, Susan; Murphy, Graham J (2006). Ursula K. Le Guin : a critical companion. Westport, Conn: Greenwood Press. p. 18. ISBN 9780313027307. OCLC 230345464.
2. Chatham, Mark (2012). Structured Query Language By Example - Volume I: Data Query Language. p. 8. ISBN 978-1-29119951-2.