Lab 10 -Connecting a Python Application to a MySQL Database

In this lab, you will learn how to connect a python application on streamlit with a MySQL

Objectives:
1. Create a CRUD Application using Streamlit
2. Connect the application to MySQL Server

database. The entire procedure is described in the following steps:

Tools required:

1. **Python:** https://www.python.org/downloads/
2. **PyCharm:** PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers. https://www.jetbrains.com/pycharm/download/#section=windows
3. **Streamlit:** It is an open source app framework in Python language. It helps us create web apps for data science and machine learning in a short time. It is compatible with major Python libraries. Command: *pip install streamlit*
4. **MySQL:** It is a widely used relational database management system (RDBMS). https://dev.mysql.com/doc/mysql-installation-excerpt/5.7/en/
5. **MySQL WorkBench**: MySQL Workbench is a unified visual tool for database architects, developers, and DBAs. https://dev.mysql.com/downloads/workbench/

Note: MySQL Workbench is a MySQL Server GUI. It requires a MySQL Server connection for most tasks. Documentation: WorkBench

Step 1: Installations

Step 2: Connecting the application to a MySQL Server

Once a new project is created in PyCharm, a file named 'app.py' is created inside the project. When the code is executed for the first time, the below snippet is executed to create a database named 'ebike'. Upon execution, the code is commented out.

app.py: The main function which can be used to start the application.

app.py:

```
import mysql.connector
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="password"
)
c = mydb.cursor()
c.execute("CREATE DATABASE ebike")
```

MySQL Connector: It enables Python programs to access MySQL databases, using an API that is compliant with the Python Database API.

mysql.connector.connect: This method sets up a connection, establishing a session with the MySQL server. If no arguments are given, it uses the already configured or default values.

mydb.cursor(): It is used to execute statements to communicate with the MySQL database.

Note: This code is later written into the file 'database.py' to connect the application to the database, a connection for the same can be created in the MySQL WorkBench.

Step 3: Writing SQL Statements for CRUD Operations

Creating a table inside ebike database:

```
CREATE TABLE IF NOT EXISTS DEALER(dealer_id TEXT, dealer_name TEXT, dealer_city
TEXT, dealer_pin TEXT, dealer_street TEXT)
```

Adding a dealer into the table:

```
INSERT INTO DEALER(dealer_id, dealer_name, dealer_city, dealer_pin, dealer_street)
VALUES (%s,%s,%s,%s,%s)
```

Viewing all the data that has been added:

```
SELECT * FROM DEALER
```

Viewing only the dealers:

```
SELECT dealer_name FROM DEALER
```

Getting a particular dealer:

```
SELECT * FROM DEALER WHERE dealer_name="{}"
```

Editing an already created dealer:

```
UPDATE DEALER SET dealer_id=%s, dealer_name=%s, dealer_city=%s, dealer_pin=%s,
dealer_street=%s WHERE dealer_id=%s and dealer_name=%s and dealer_city=%s and
dealer_pin=%s and dealer_street=%s
```

Deleting a dealer:

```
DELETE FROM DEALER WHERE dealer_name="{}"
```

Step 4: Creating the Python Application for CRUD Operations

Here, we will try to execute an eBike Dealer Application where we can create, view, update and delete the dealers. Code is briefly explained.

Link to code: eBike Application

Command to run the application: *streamlit run <path of the file>*

4.A: The **app.py** file acts as the main function that calls other functions like **create()**, **read()**, **update()** and **delete()** which have been written as separate files for clarity.

app.py:
```
import streamlit as st
from create import create
from database import create_table
from delete import delete
from read import read
from update import update
def main():
    st.title("eBike App")
    menu = ["Add", "View", "Edit", "Remove"]
    choice = st.sidebar.selectbox("Menu", menu)
    create_table()
    if choice == "Add":
        st.subheader("Enter Dealer Details:")
        create()
    elif choice == "View":
        st.subheader("View created tasks")
        read()
    elif choice == "Edit":
        st.subheader("Update created tasks")
        update()
    elif choice == "Remove":
        st.subheader("Delete created tasks")
        delete()
    else:
        st.subheader("About tasks")
if __name__ == '__main__':
```

4.B: Code for create.py, read.py, update.py and delete.py

- **create.py:** Here, a dealer is created and added to the database from the UI. Streamlit provides various user friendly functions like columns, selectbox etc which can be used to

create.py:
```

import streamlit as st
from database import add_data

def create():
    col1, col2 = st.columns(2)
    with col1:
        dealer_id = st.text_input("ID:")
        dealer_name = st.text_input("Name:")
    with col2:
        dealer_city = st.selectbox("City", ["Bangalore", "Chennai", "Mumbai"])
        dealer_pin = st.text_input("Pin Code:")
    dealer_street = st.text_input("Street Name:")
    if st.button("Add Dealer"):
        add_data(dealer_id, dealer_name, dealer_city, dealer_pin, dealer_street)
        st.success("Successfully added dealer: {}".format(dealer_name))
```

create an interactive UI. The records of the dealer can be visualised in MySQL WorkBench.

- **read.py:** In this file, you can view all the dealers added in the UI and also use plotly to visualise the locations of the dealers added in the form of a pie chart.

read.py:

```
import pandas as pd
import streamlit as st
import plotly.express as px
from database import view_all_data

def read():
    result = view_all_data()
    # st.write(result)
    df = pd.DataFrame(result, columns=['Dealer ID', 'Dealer Name', 'Dealer City', 'Dealer Pin',
'Dealer Street'])
    with st.expander("View all Dealers"):
        st.dataframe(df)
    with st.expander("Dealer Location"):
        task_df = df['Dealer City'].value_counts().to_frame()
        task_df = task_df.reset_index()
        st.dataframe(task_df)
        p1 = px.pie(task_df, names='index', values='Dealer City')
```

update.py:

```
import pandas as pd
import streamlit as st
from database import view_all_data, view_only_dealer_names, get_dealer, edit_dealer_data

def update():
    result = view_all_data()
    df = pd.DataFrame(result, columns=['Dealer ID', 'Dealer Name', 'Dealer City', 'Dealer Pin',
'Dealer Street'])
    with st.expander("Current Dealers"):
        st.dataframe(df)
    list_of_dealers = [i[0] for i in view_only_dealer_names()]
    selected_dealer = st.selectbox("Dealer to Edit", list_of_dealers)
    selected_result = get_dealer(selected_dealer)
    if selected_result:
        dealer_id = selected_result[0][0]
        dealer_name = selected_result[0][1]
        dealer_city = selected_result[0][2]
        dealer_pin = selected_result[0][3]
        dealer_street = selected_result[0][4]

        col1, col2 = st.columns(2)
        with col1:
            new_dealer_id = st.text_input("ID:", dealer_id)
            new_dealer_name = st.text_input("Name:", dealer_name)
        with col2:
            new_dealer_city = st.selectbox(dealer_city, ["Bangalore", "Chennai", "Mumbai"])
            new_dealer_pin = st.text_input("Pin Code:", dealer_pin)
        new_dealer_street = st.text_input("Street Name:", dealer_street)
        if st.button("Update Dealer"):
            edit_dealer_data(new_dealer_id, new_dealer_name, new_dealer_city, new_dealer_pin,
new_dealer_street, dealer_id, dealer_name, dealer_city, dealer_pin, dealer_street)
            st.success("Successfully updated:: {} to ::{}".format(dealer_name, new_dealer_name))
    result2 = view_all_data()
    df2 = pd.DataFrame(result2, columns=['Dealer ID', 'Dealer Name', 'Dealer City', 'Dealer Pin',
```

- **update.py:** In this file, you can update the details of a selected dealer that already exists in the database and see the changes in the UI itself.

- **delete.py:** Deleting the record of a selected dealer that already exists in the database. Upon

delete.py:

```
import pandas as pd
import streamlit as st
from database import view_all_data, view_only_dealer_names, delete_data

def delete():
    result = view_all_data()
    df = pd.DataFrame(result, columns=['Dealer ID', 'Dealer Name', 'Dealer City', 'Dealer Pin',
'Dealer Street'])
    with st.expander("Current data"):
        st.dataframe(df)
    list_of_dealers = [i[0] for i in view_only_dealer_names()]
    selected_dealer = st.selectbox("Task to Delete", list_of_dealers)
    st.warning("Do you want to delete ::{}".format(selected_dealer))
    if st.button("Delete Dealer"):
        delete_data(selected_dealer)
        st.success("Dealer has been deleted successfully")
    new_result = view_all_data()
    df2 = pd.DataFrame(new_result, columns=['Dealer ID', 'Dealer Name', 'Dealer City', 'Dealer
Pin', 'Dealer Street'])
    with st.expander("Updated data"):
        st.dataframe(df2)
```

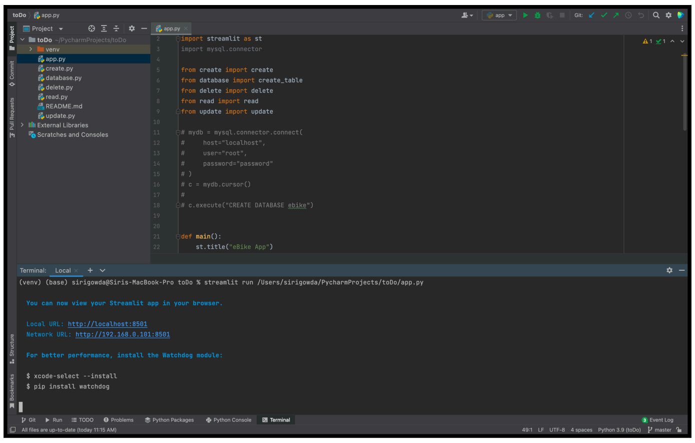
deleting a dealer in the UI, the same can be seen in the SQL Database from MySQL WorkBench.

database.py:

```
import mysql.connector
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="password",
    database="ebike"
)
c = mydb.cursor()

def create_table():
    c.execute('CREATE TABLE IF NOT EXISTS DEALER(dealer_id TEXT, dealer_name TEXT,
dealer_city TEXT, dealer_pin TEXT, dealer_street TEXT)')

def add_data(dealer_id, dealer_name, dealer_city, dealer_pin, dealer_street):
    c.execute('INSERT INTO DEALER(dealer_id, dealer_name, dealer_city, dealer_pin,
dealer_street) VALUES (%s,%s,%s,%s,%s)',(dealer_id, dealer_name, dealer_city, dealer_pin,
dealer_street))
    mydb.commit()

def view_all_data():
    c.execute('SELECT * FROM DEALER')
    data = c.fetchall()
    return data

def view_only_dealer_names():
    c.execute('SELECT dealer_name FROM DEALER')
    data = c.fetchall()
    return data

def get_dealer(dealer_name):
    c.execute('SELECT * FROM DEALER WHERE dealer_name="{}"'.format(dealer_name))
    data = c.fetchall()
    return data

def edit_dealer_data(new_dealer_id, new_dealer_name, new_dealer_city, new_dealer_pin,
new_dealer_street, dealer_id, dealer_name, dealer_city, dealer_pin, dealer_street):
    c.execute("UPDATE DEALER SET dealer_id=%s, dealer_name=%s, dealer_city=%s,
dealer_pin=%s, dealer_street=%s WHERE dealer_id=%s and dealer_name=%s and
dealer_city=%s and dealer_pin=%s and dealer_street=%s", (new_dealer_id, new_dealer_name,
new_dealer_city, new_dealer_pin, new_dealer_street, dealer_id, dealer_name, dealer_city,
dealer_pin, dealer_street))
    mydb.commit()
    data = c.fetchall()
    return data

def delete_data(dealer_name):
    c.execute('DELETE FROM DEALER WHERE dealer_name="{}"'.format(dealer_name))
    mydb.commit()
```

4.C: **database.py:** In this file, you make a connection between the server and the python application. The above SQL statements are implemented within this file.

Running the



```
import streamlit as st
import mysql.connector

from create import create
from database import create_table
from delete import delete
from read import read
from update import update

# mydb = mysql.connector.connect(
#     host="localhost",
#     user="root",
#     password="password"
# )
# c = mydb.cursor()
#
# c.execute("CREATE DATABASE ebike")


def main():
    st.title("eBike App")
```

```
(venv) (base) sirigowda@Siris-MacBook-Pro toDo % streamlit run /Users/sirigowda/PycharmProjects/toDo/app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.0.101:8501

  For better performance, install the Watchdog module:

  $ xcode-select --install
  $ pip install watchdog
```

Application:

Assignment: Railway Reservation

1. Execute a CRUD (Create, Read, Update and Delete) application in python using Streamlit and MySQL to create a table 'train' in the User-Interface.

Commented [1]:

The 'train' table should be populated with the following 3 records using the User-Interface:

Train_No	**Name**	**Train_Type**	**Source**	**Destination**	**Availability**
62621	BEN-CHE Shatabdi	Superfast	Bengaluru	Chennai	yes
62620	CHE-BEN Shatabdi	Fast	Chennai	Bengaluru	No
25261	Managaluru Mail	Mail	Chennai	Mangaluru	Yes

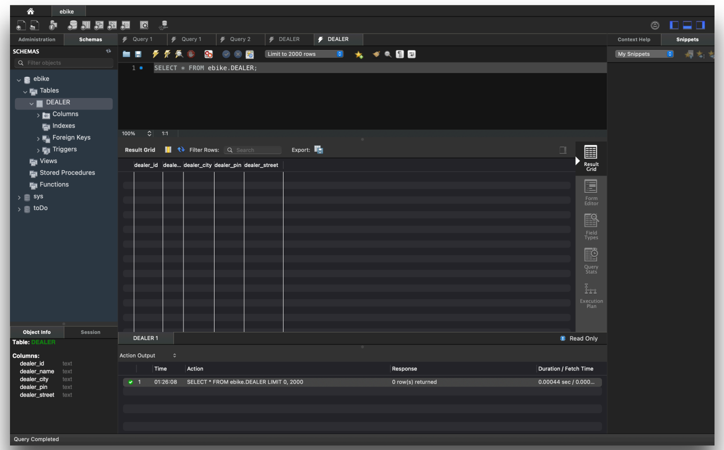
2. Read the details entered at real time in the User-Interface itself.
3. Update the 'Availability' of the Train_No 62620 to 'yes' in the User-Interface.
4. Delete the Train_No 25261 in the User-Interface.

Deliverables for submission: Upload a PDF with the following 6 Screenshots. (Remember to incorporate your SRN)

1. Screenshot of database with the table - 'train' before populating it.
2. Screenshot of the User Interface.

3. Screenshot of the 3 records in the train table from MySQL WorkBench.
4. Screenshot of the same 3 records visualised in the User Interface.
5. Screenshot of Updated Train_No 62620 in the User-Interface.
6. Screenshot of User-Interface after the Train_No 25261 has been deleted.

Sample



```
SELECT * FROM ebike.DEALER;
```

Table: DEALER

Columns:

dealer_id	text
dealer_name	text
dealer_city	text
dealer_pin	text
dealer_street	text

Result Grid columns: dealer_id, deale..., dealer_city, dealer_pin, dealer_street

Action Output

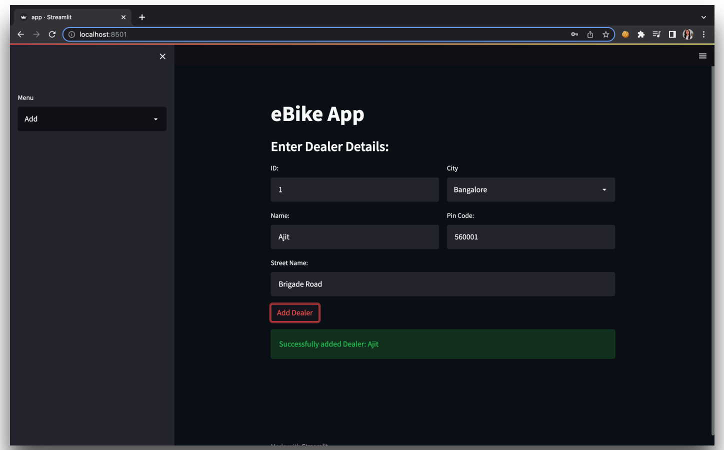
	Time	Action	Response	Duration / Fetch Time
✓ 1	01:26:08	SELECT * FROM ebike.DEALER LIMIT 0, 2000	0 row(s) returned	0.00044 sec / 0.000...

Query Completed

screenshots as per demo exercise:

1.

2.



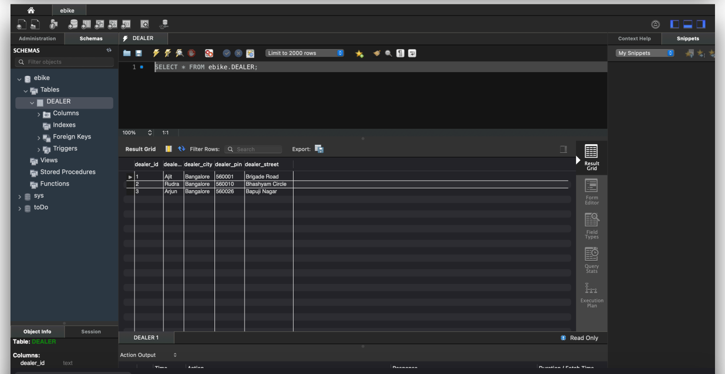
eBike App

Enter Dealer Details:

ID: 1	City: Bangalore
Name: Ajit	Pin Code: 560001
Street Name: Brigade Road	

Successfully added Dealer: Ajit

3.

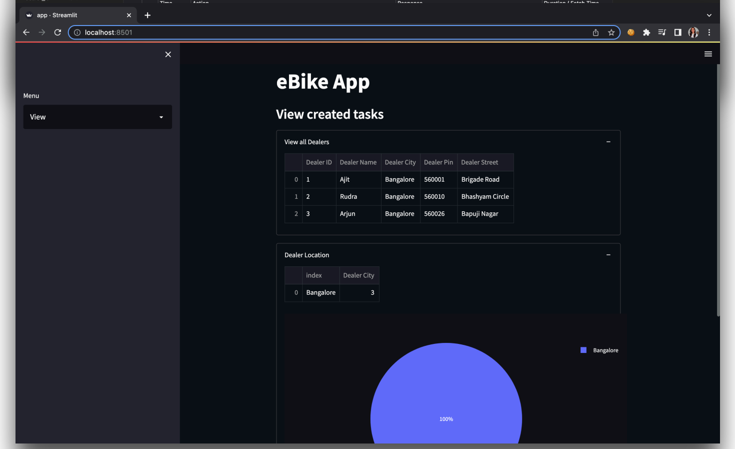


```
SELECT * FROM ebike.DEALER;
```

dealer_id	deale...	dealer_city	dealer_pin	dealer_street
1	Ajit	Bangalore	560001	Brigade Road
2	Rudra	Bangalore	560010	Bhashyam Circle
3	Arjun	Bangalore	560026	Bapuji Nagar

Table: DEALER
Columns:
dealer_id text

4.



eBike App

View created tasks

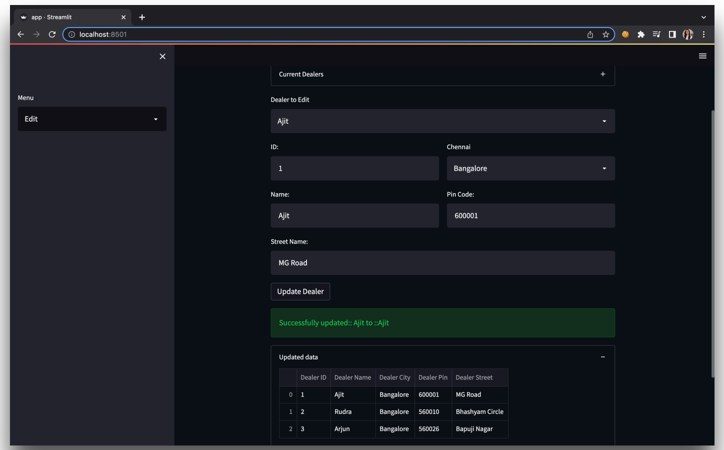
View all Dealers

	Dealer ID	Dealer Name	Dealer City	Dealer Pin	Dealer Street
0	1	Ajit	Bangalore	560001	Brigade Road
1	2	Rudra	Bangalore	560010	Bhashyam Circle
2	3	Arjun	Bangalore	560026	Bapuji Nagar

Dealer Location

	index	Dealer City
0	Bangalore	3

5.



Dealer to Edit: Ajit

ID: 1

Chennai: Bangalore

Name: Ajit

Pin Code: 600001

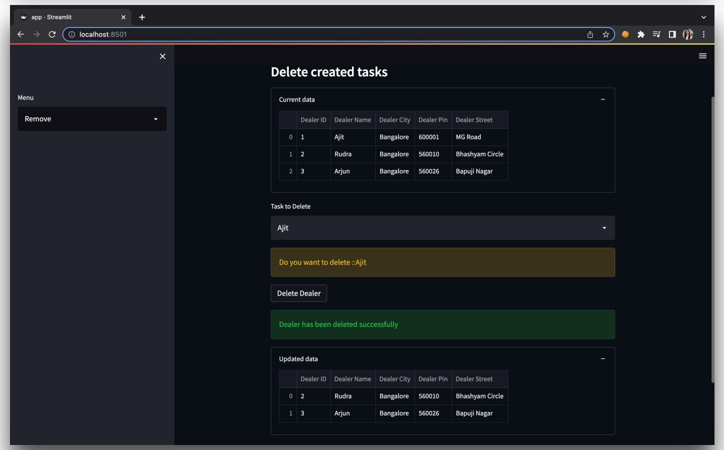
Street Name: MG Road

Successfully updated:: Ajit to ::Ajit

Updated data

	Dealer ID	Dealer Name	Dealer City	Dealer Pin	Dealer Street
0	1	Ajit	Bangalore	600001	MG Road
1	2	Rudra	Bangalore	560010	Bhashyam Circle
2	3	Arjun	Bangalore	560026	Bapuji Nagar

6.



Delete created tasks

Current data

	Dealer ID	Dealer Name	Dealer City	Dealer Pin	Dealer Street
0	1	Ajit	Bangalore	600001	MG Road
1	2	Rudra	Bangalore	560010	Bhashyam Circle
2	3	Arjun	Bangalore	560026	Bapuji Nagar

Task to Delete: Ajit

Do you want to delete ::Ajit

Dealer has been deleted successfully

Updated data

	Dealer ID	Dealer Name	Dealer City	Dealer Pin	Dealer Street
0	2	Rudra	Bangalore	560010	Bhashyam Circle
1	3	Arjun	Bangalore	560026	Bapuji Nagar