

# SWE 642 EXTRA CREDIT REPORT

## Team Members:

Jithendra Sai Pappuri – G01506453 - [jpappuri@gmu.edu](mailto:jpappuri@gmu.edu)

Venkata Abhiram Karuturi – G01505660 – [vkarutur@gmu.edu](mailto:vkarutur@gmu.edu)

Ganesh Jasti – G01505410 – [gjasti@gmu.edu](mailto:gjasti@gmu.edu)

Lakshmi Sankari Vissapragada - G01481312- [lvissapr@gmu.edu](mailto:lvissapr@gmu.edu)

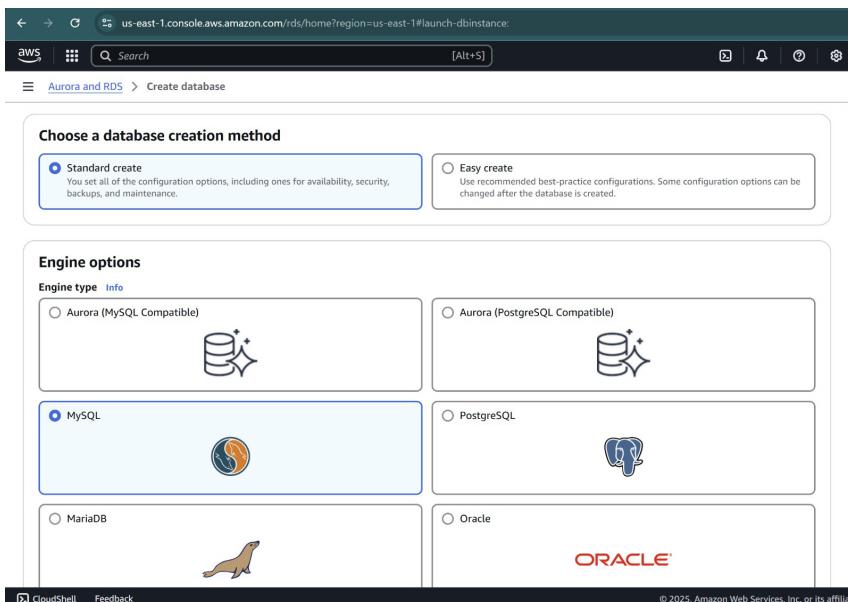
## Technologies Used:

1. Django
2. Amazon RDS DB
3. MySQL

## Project Setup:

### Creating MySQL DB using Amazon RDS:

1. Login to AWS console and go to RDS section.
2. Create a Database
3. Select Standard Create from the given database create options.
4. Select MySQL from the given database options.



5. leave the engine options as it is.
6. Select **Free Tier** from the templates.

**Templates**

Choose a sample template to meet your use case.

- Production**  
Use defaults for high availability and fast, consistent performance.
- Dev/Test**  
This instance is intended for development use outside of a production environment.
- Free tier**  
Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS. [Info](#)

**Availability and durability**

**Deployment options** [Info](#)

Choose the deployment option that provides the availability and durability needed for your use case. AWS is committed to a certain level of uptime depending on the deployment option you choose. Learn more in the [Amazon RDS service level agreement \(SLA\)](#).

- Multi-AZ DB cluster deployment (3 instances)  
Creates a primary DB instance with two readable standbys in separate Availability Zones. This setup provides:
  - 99.95% uptime
  - Redundancy across Availability Zones
  - Increased read capacity
  - Reduced write latency
- Multi-AZ DB instance deployment (2 instances)  
Creates a primary DB instance with a non-readable standby instance in a separate Availability Zone. This setup provides:
  - 99.95% uptime
  - Redundancy across Availability Zones
- Single-AZ DB instance deployment (1 instance)  
Creates a single DB instance without standby instances. This setup provides:
  - 99.5% uptime
  - No data redundancy

© 2025, Amazon Web Services, Inc. or its affiliates

7. Select a name for database and credentials for username and password.
8. Select **db.t3.micro** in Instance Configuration.

**Instance configuration**

The DB instance configuration options below are limited to those supported by the engine that you selected above.

**DB instance class** [Info](#)

▼ Hide filters

Show instance classes that support Amazon RDS Optimized Writes

**Info**  
Amazon RDS Optimized Writes improves write throughput by up to 2x at no additional cost.

Include previous generation classes

Standard classes (includes m classes)

Memory optimized classes (includes r and x classes)

**burstable classes (includes t classes)**

**db.t3.micro**

2 vCPUs   1 GiB RAM   Network: Up to 2,085 Mbps

9. Leave the storage option as it is.

The screenshot shows the 'Storage' configuration section of the AWS Aurora and RDS 'Create database' wizard. It includes fields for 'Storage type' (set to 'General Purpose SSD (gp2)'), 'Allocated storage' (set to 20 GiB), and a link to 'Additional storage configuration'.

10. Don't change anything in **Connectivity** and allow the **Public Access**.

The screenshots show the 'Connectivity' and 'Public access' configuration sections. In 'Connectivity', the 'Compute resource' section has 'Don't connect to an EC2 compute resource' selected. The 'Virtual private cloud (VPC)' section shows 'Default VPC (vpc-0cb17b4b2b63fbaf)' selected. A note says 'After a database is created, you can't change its VPC.' In the 'Public access' section, 'Yes' is selected, allowing public access to the database.

11. Leave the rest of the database settings as it is and scroll to the end and select **Create Database**.

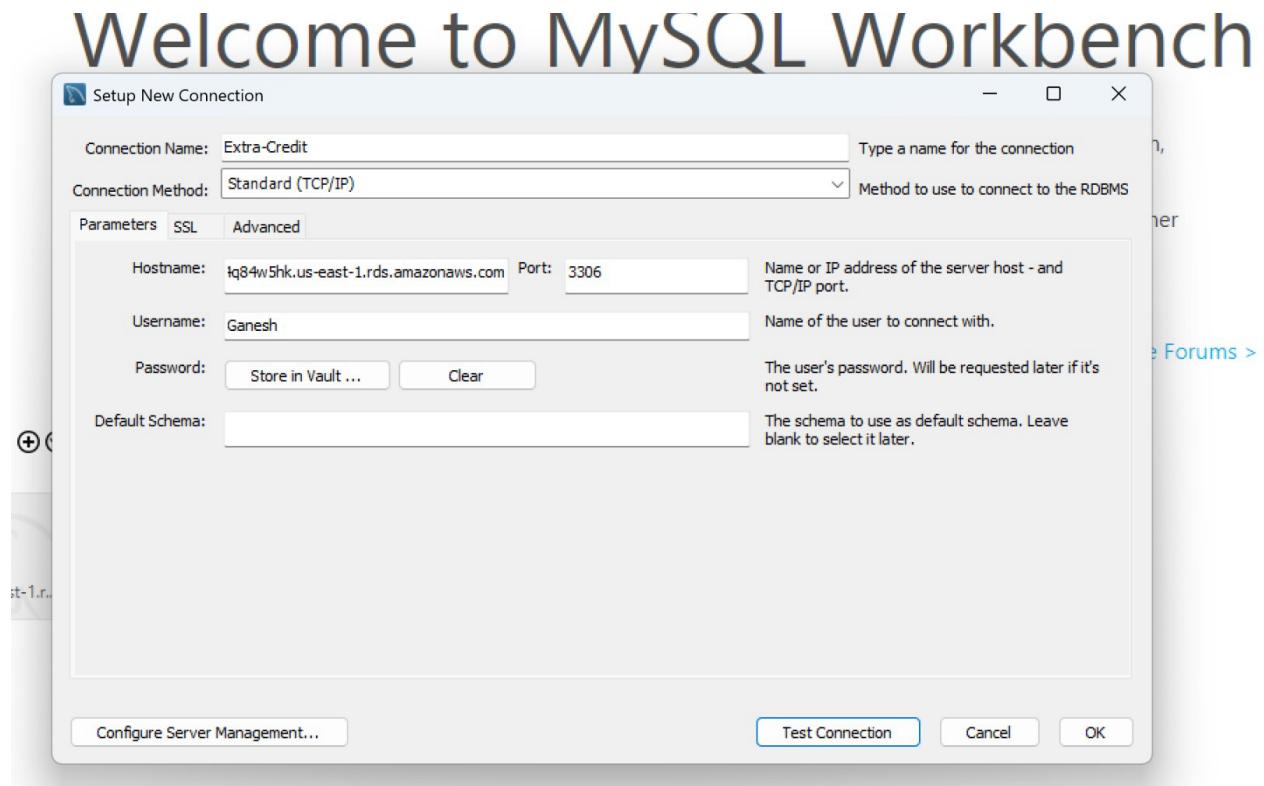
12. The database will be created.

13. After creating, we will get a Database endpoint.

**DB Endpoint:** database-1.ckxy4q84w5hk.us-east-1.rds.amazonaws.com

### Using MySQL Workbench to connect to DB:

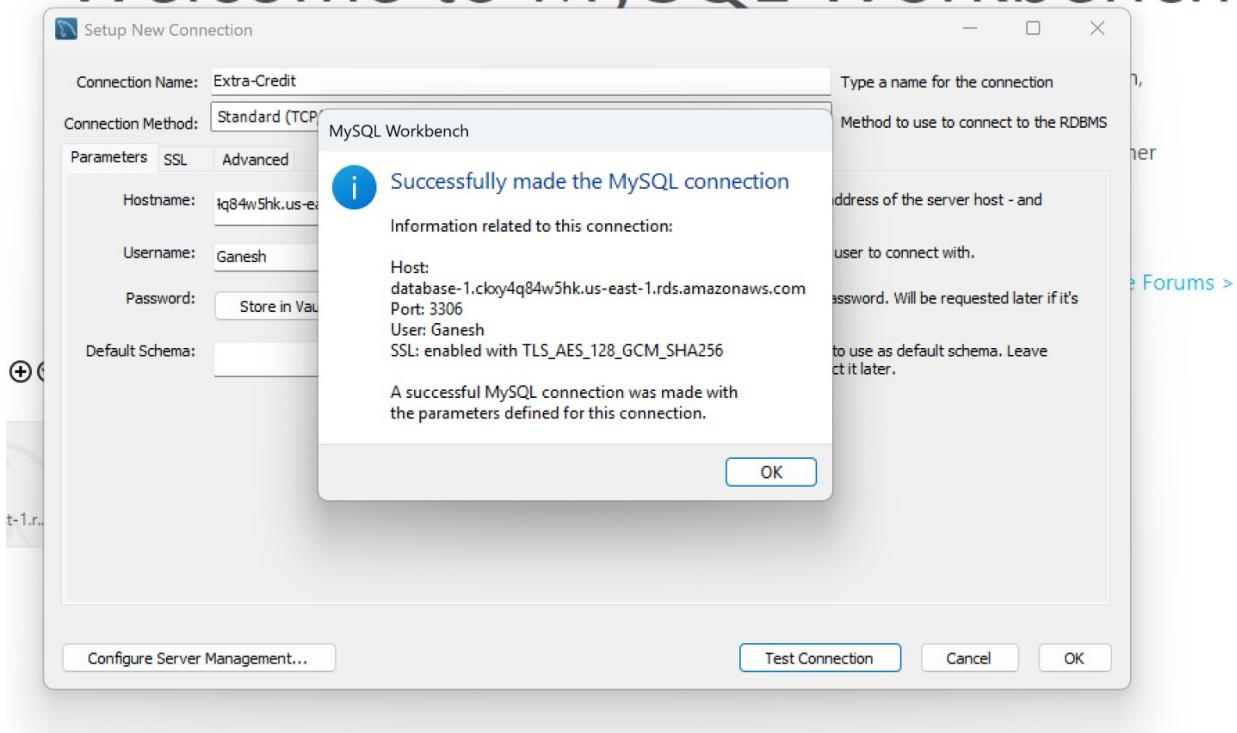
1. To create a new connection, select the + icon in the MySQL Workbench
2. Enter connection name to database.
3. enter the endpoint **database-1.ckxy4q84w5hk.us-east-1.rds.amazonaws.com** as host.
4. Enter username and password that was given while creating AWS database.



5. Click test connection and check if the connection is established.

6. If established successfully you will get below message.

# Welcome to MySQL Workbench



7. Now the connection was established to MySQL.
8. Create a database with name surveys using the command in workbench

```
create database survey_details;
```

## Creating Django Project(front end and backend):

For this assignment we are using python framework Django for both front end and back end.

1. First create a new folder in Visual studio and name it Django-copy.
2. Install Django in the above folder using the below command:

```
pip install django
```

3. After installing django we will see some default files and folders like models.py, views.py, forms.py, manage.py
4. Now create a subfolder named **survey\_app** for frontend and **survey\_project** for backend.

5. Now create a subfolder named templates in survey\_app and create the below files

base.html

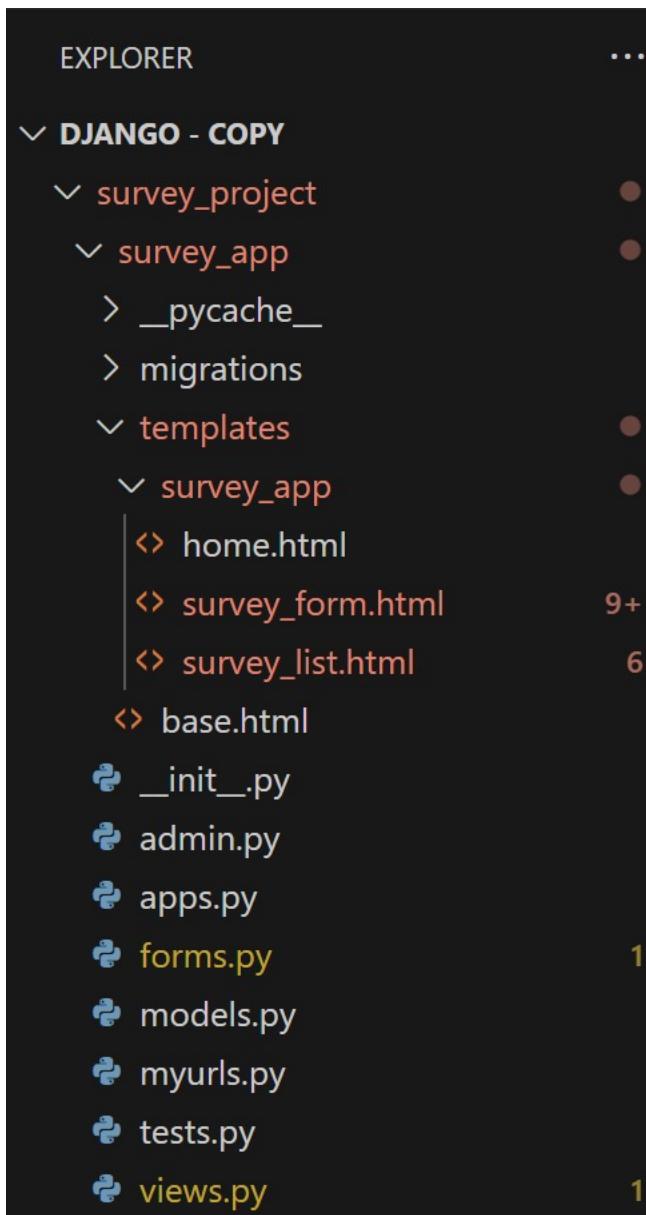
home.html

survey\_form.html

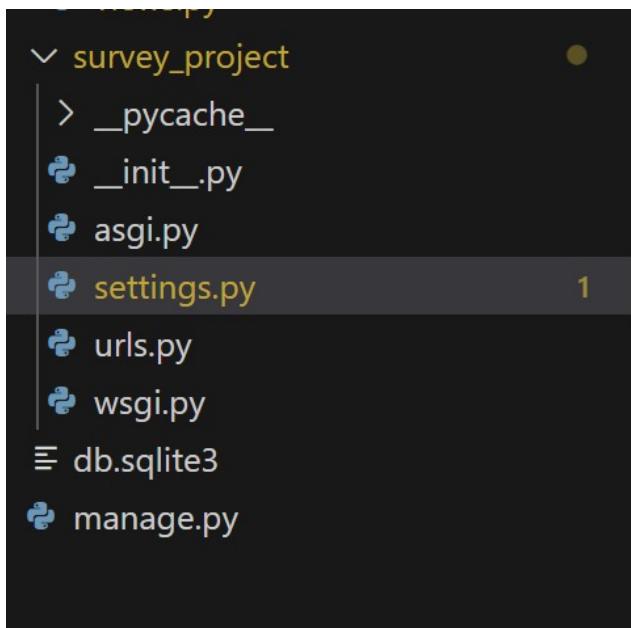
survey\_list.html

6. Below is the structure of the project

### Front end:



## Backend:



7. For backend database connectivity edit setting.py file and add the respective database details as shown in the below screenshot.

```
survey_project > survey_project > settings.py > ...
73 # Database
74 # https://docs.djangoproject.com/en/5.2/ref/settings/#databases
75
76 DATABASES = {
77     'default': {
78         'ENGINE': 'django.db.backends.mysql',
79         'NAME': 'survey_details',
80         'USER': 'Ganesh',
81         'PASSWORD': 'Gangstar_2025',
82         'HOST': 'database-1.ckxy4q84w5hk.us-east-1.rds.amazonaws.com',
83         'PORT': '3306',
84         'OPTIONS': {
85             'init_command': "SET sql_mode='STRICT_TRANS_TABLES'",
86         },
87     }
88 }
89
90
```

## **Implementation details:**

### **1. Model Design (models.py)**

The Survey model was created to store survey information. Each field in the model represents one form input, such as:

`first_name`, `last_name`, `email`, etc. – simple text fields

`liked_most` – a MultiSelectField (or similar logic) to allow multiple choices

`source_of_interest` – a single choice field using Django choices

`date_of_survey` – stores the selected date

Each field corresponds to a form input (e.g., first name, liked options, etc.).

### **2. Form Handling (forms.py)**

A SurveyForm (a subclass of ModelForm) was implemented to automatically generate the form based on the Survey model fields. Custom widgets and validation rules are handled here. In Django, custom widgets are used to control how specific form fields are rendered in HTML.

Eg: The `date_of_survey` field uses a DateInput widget with `type="date"` so that it shows a proper calendar date picker in the browser.

### **3. View Logic (views.py)**

#### **`survey_form_view()`**

Handles both creation and update based on the presence of an id.

On valid submission, saves the data and uses Django's messages to show feedback.

If updating, it redirects to the survey list page to give users a confirmation via alert.

#### **`survey_list_view()`**

Fetches all survey objects and displays them in a table.

#### **`delete_survey_view()`**

Deletes a survey after confirmation.

Shows a “Deleted successfully” alert using Django messages and JavaScript alert()

## **4. Templates (.html Files)**

### **survey\_form.html:**

Renders the form with Bootstrap styling and JavaScript zip code autofill.

### **survey\_list.html:**

Displays all surveys and provides "**Update**" and "**Delete**" actions for each.

### **base.html:**

Shared layout template.

## **5. Message alerts for form submissions and actions:**

Django's messages.success() is used to pass messages like:

"Survey submitted successfully!"

"Survey updated successfully!"

"Deleted successfully!"

These messages are rendered in JavaScript alert boxes for immediate feedback

## **6. Database Integration**

MySQL is configured in settings.py to store data remotely.

Migrations (makemigrations and migrate) are used to sync models to tables.

Data retrieval and manual queries are tested via MySQL Workbench.

## **Running the project:**

Save the project. Now enter the below commands.

**python manage.py makemigrations**

**python manage.py migrate**

**python manage.py runserver 8002**

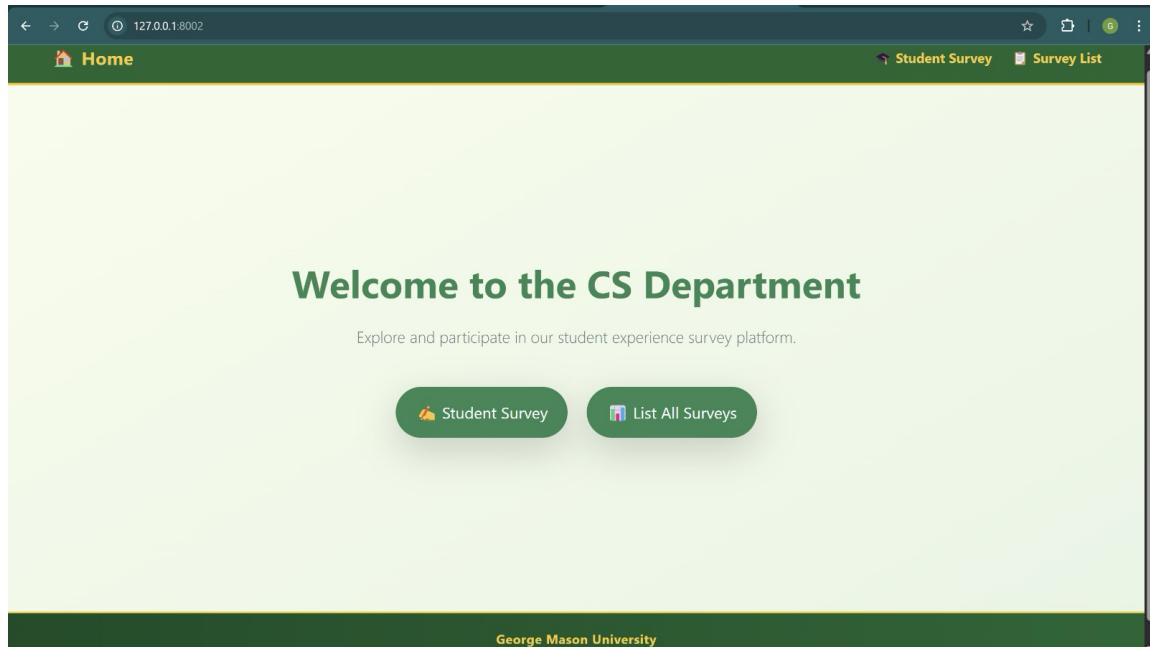
```
PROBLEMS 30 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\96036\Documents\SWE_642\ Django - Copy\survey_project> python manage.py makemigrations
>>
No changes detected
● PS C:\Users\96036\Documents\SWE_642\ Django - Copy\survey_project> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, survey_app
Running migrations:
  No migrations to apply.
PS C:\Users\96036\Documents\SWE_642\ Django - Copy\survey_project> python manage.py runserver 8002
Watching for file changes with StatReloader
Performing system checks...
System check identified no issues (0 silenced).
May 05, 2025 - 23:26:15
Django version 5.2, using settings 'survey_project.settings'
Starting development server at http://127.0.0.1:8002/
Quit the server with CTRL-BREAK.

WARNING: This is a development server. Do not use it in a production setting. Use a production WSGI or ASGI server instead.
For more information on production servers see: https://docs.djangoproject.com/en/5.2/howto/deployment/
```

Now we have the application running in localhost on port 8002.

## Screenshots of front end pages:

### Home Page:



## Student survey page:

The screenshot shows a web-based survey form titled "CS Department Survey". The form is divided into several sections:

- Personal Information:** Fields for First Name, Last Name, Email, Street Address, Zip Code, City, State, and Telephone.
- Campus Preferences:** A list of checkboxes for "Students", "Location", "Campus", "Atmosphere", "Dorm Rooms", and "Sports".
- Interest in University:** A list of radio buttons for "Friends", "Television", "Internet", and "Other".
- Additional Comments:** A text area for "Enter your feedback".
- Likelihood of Recommending:** A dropdown menu set to "Select".
- Action Buttons:** "Submit" (green button) and "Reset" (yellow button).

The URL in the browser bar is 127.0.0.1:8002/survey/.

## Survey list:

127.0.0.1:8002/surveys/

| ID | First Name | Last Name | Email                     | Street Address             | Zip   | City    | State | Phone       | Date         | Liked Most              | Interest Source | Comments | Recommendation | Actions  |
|----|------------|-----------|---------------------------|----------------------------|-------|---------|-------|-------------|--------------|-------------------------|-----------------|----------|----------------|--|
| 1  | Ganesh     | Jasti     | gjasti@gmu.edu            | 4307 - Ramona Drive, Apt J | 22030 | Fairfax | VA    | 5719109089  | May 3, 2025  | ['atmosphere', 'dorms'] |                 |          |                | <button>Update</button><br><button>Delete</button> |
| 2  | Ganesh     | Jasti     | gjasti@gmu.edu            | 4307 - Ramona Drive, Apt J | 22030 | Fairfax | VA    | 09573542991 | May 29, 2025 | []                      |                 |          |                | <button>Update</button><br><button>Delete</button> |
| 3  | Ganesh     | Jasti     | gjasti@gmu.edu            | 4307 - Ramona Drive, Apt J | 22030 | Fairfax | VA    | 5719109089  | May 15, 2025 | []                      |                 |          |                | <button>Update</button><br><button>Delete</button> |
| 6  | Ganesh     | Jasti     | gjasti@gmu.edu            | 4307 - Ramona Drive, Apt J | 22030 | Fairfax | VA    | 5719109089  | May 3, 2025  | ['students']            | tv              |          |                | <button>Update</button><br><button>Delete</button> |
| 9  | Ganesh     | Jasti     | ganeshjasti0912@gmail.com | 4307 Ramona Drive          | 22030 | Fairfax | VA    | 5719109089  | June 3, 2025 | ['dorms', 'sports']     |                 |          |                | <button>Update</button><br><button>Delete</button> |

## Deleting record :

127.0.0.1:8002/surveys/

127.0.0.1:8002 says  
Are you sure you want to delete this survey?

| ID | First Name | Last Name | Email                     | Street Address             | Zip   | City    | State | Phone       | Date         | Liked Most              | Interest Source | Comments | Recommendation | Actions  |
|----|------------|-----------|---------------------------|----------------------------|-------|---------|-------|-------------|--------------|-------------------------|-----------------|----------|----------------|--|
| 1  | Ganesh     | Jasti     | gjasti@gmu.edu            | 4307 - Ramona Drive, Apt J | 22030 | Fairfax | VA    | 5719109089  | May 3, 2025  | ['atmosphere', 'dorms'] |                 |          |                | <button>Update</button><br><button>Delete</button> |
| 2  | Ganesh     | Jasti     | gjasti@gmu.edu            | 4307 - Ramona Drive, Apt J | 22030 | Fairfax | VA    | 09573542991 | May 29, 2025 | []                      |                 |          |                | <button>Update</button><br><button>Delete</button> |
| 3  | Ganesh     | Jasti     | gjasti@gmu.edu            | 4307 - Ramona Drive, Apt J | 22030 | Fairfax | VA    | 5719109089  | May 15, 2025 | []                      |                 |          |                | <button>Update</button><br><button>Delete</button> |
| 6  | Ganesh     | Jasti     | gjasti@gmu.edu            | 4307 - Ramona Drive, Apt J | 22030 | Fairfax | VA    | 5719109089  | May 3, 2025  | ['students']            | tv              |          |                | <button>Update</button><br><button>Delete</button> |
| 9  | Ganesh     | Jasti     | ganeshjasti0912@gmail.com | 4307 Ramona Drive          | 22030 | Fairfax | VA    | 5719109089  | June 3, 2025 | ['dorms', 'sports']     |                 |          |                | <button>Update</button><br><button>Delete</button> |

## Database result:

The screenshot shows the MySQL Workbench interface with the following details:

- Query Editor:** Displays the following SQL queries:
  - CREATE DATABASE survey\_details;
  - SHOW TABLES FROM survey\_details;
  - USE survey\_details;
  - SELECT \* FROM survey\_app\_surveys
- Result Grid:** Shows the data from the survey\_app\_surveys table. The columns are: id, first\_name, last\_name, address, city, state, zip, phone, email, and date\_c. The data includes 15 rows of student information.
- Output Panel:** Shows the execution history with two entries:

| # | Time     | Action  | Message            | Duration / Fetch      |
|---|----------|---|--------------------|-----------------------|
| 1 | 21:45:07 | USE survey_details                            | 0 row(s) affected  | 0.016 sec             |
| 2 | 21:45:07 | SELECT * FROM survey_app_survey LIMIT 0, 1000 | 10 row(s) returned | 0.015 sec / 0.000 sec |

## Conclusion:

We were able to implement CRUD operation to manage student survey data in the database such as MySQL successfully by implementing the both frontend application and back end application using Django.