Database Schema Overview

1. **Table:** `departments`
   - **Purpose:** Represents university departments.
   - **Fields:**
     - `id`: Primary key, uniquely identifies a department.
     - `name`: Name of the department (unique).
     - `code`: A unique, maximum four-character code representing the department.
   - **Relationships:** Has many `Faculty` members and `Programs`.
2. **Table:** `faculty`
   - **Purpose:** Stores information about faculty members.
   - **Fields:**
     - `id`: Primary key, uniquely identifies a faculty member.
     - `name`: Name of the faculty member.
     - `email`: Email address of the faculty member (unique).
     - `rank`: Academic rank (full, associate, assistant, adjunct).
     - `department_code`: Foreign key linking to the `departments` table.
   - **Relationships:** Belongs to a `Department`.
3. **Table:** `programs`
   - **Purpose:** Represents academic programs or degrees.
   - **Fields:**
     - `id`: Primary key, uniquely identifies a program.
     - `name`: Name of the program (unique).
     - `department_code`: Foreign key linking to the `departments` table.
     - `in_charge_id`: Foreign key linking to a `Faculty` member who is in charge of the program.
   - **Relationships:** Belongs to a `Department` and has many `Courses` through `program_courses`.
4. **Table:** `courses`
   - **Purpose:** Contains information about courses offered by departments.
   - **Fields:**
     - `id`: Primary key, a combination of department code and a 4-digit number.
     - `title`: Title of the course.
     - `description`: A text describing the course.
     - `department_code`: Foreign key linking to the `departments` table.
   - **Relationships:** Belongs to a `Department`, has many `Sections`, and is associated with many `Programs` through `program_courses`.
5. **Table:** `sections`
   - **Purpose:** Details about specific course offerings each semester.
   - **Fields:**
     - `id`: Primary key, uniquely identifies a section.
     - `number`: A 3-digit number representing the section.
     - `semester`: The semester when the course is offered.
     - `year`: The year when the course is offered.

  - `course_id`: Foreign key linking to the `courses` table.
  - `instructor_id`: Foreign key linking to the `faculty` table.
  - `enrollment_count`: Number of students enrolled in the section.
- **Relationships:** Belongs to a `Course` and has a `Faculty` member as the instructor.

6. **Table:** `learning_objectives`
   - **Purpose:** Stores learning objectives for program evaluation.
   - **Fields:**
     - `id`: Primary key, a code for the learning objective.
     - `description`: Description of the learning objective.
     - `parent_id`: Foreign key to itself, indicating sub-objectives.
   - **Relationships:** Has many sub-objectives linking to itself.
7. **Table:** `program_courses`
   - **Purpose:** A junction table to establish many-to-many relationships between programs and courses.
   - **Fields:**
     - `program_id`: Foreign key linking to the `programs` table.
     - `course_id`: Foreign key linking to the `courses` table.
   - **Relationships:** Links `Programs` to `Courses`.
8. **Table:** `course_objectives`
   - **Purpose:** Associates courses with their respective learning objectives.
   - **Fields:**
     - `course_id`: Foreign key linking to the `courses` table.
     - `objective_id`: Foreign key linking to the `learning_objectives` table.
     - `program_id`: Foreign key linking to the `programs` table.
   - **Relationships:** Links `Courses` to `Learning Objectives`.
9. **Table:** `section_evaluations`
   - **Purpose:** Records the evaluation of learning objectives for each course section.
   - **Fields:**
     - `section_id`: Foreign key linking to the `sections` table.
     - `objective_id`: Foreign key linking to the `learning_objectives` table.
     - `evaluation_method`: The method used for evaluating the objective.
     - `students_met`: Number of students who met the objective.
   - **Relationships:** Links `Sections` to `Learning Objectives`.

Installation

Before installing the University Program Evaluation System, ensure that your environment meets the following prerequisites.

0.1 Prerequisites

- Python 3.8 or later
- Pip (Python package installer)
- A database system supported by SQLAlchemy (e.g., SQLite, PostgreSQL, MySQL)
- Tkinter for the graphical user interface (usually comes pre-installed with Python)

0.2 Setting up the Environment

1. Install Python and Pip and follow the instructions for your operating system.
2. Install required Python libraries:

   ```
   pip install sqlalchemy tk
   ```

3. Install a database system of your choice. This guide will use SQLite for simplicity.

   ```
   pip install sqlite
   ```

0.3 Downloading and Configuring the Application

1. Download the application source code from the provided repository or location.
2. Extract the contents if in a compressed format.
3. Navigate to the application's root directory in your terminal.
4. Modify the 'main.py' file to set up the database connection. Here's an example using SQLite:

   ```python
   # main.py
   from sqlalchemy import create_engine
   engine = create_engine('sqlite:///university_programs.db')
   ```

Usage

This section covers how to run and use the University Program Evaluation System.

0.4 Starting the Application

1. Open a terminal.
2. Navigate to the application's root directory.
3. Run the following command:

   ```
   python gui.py
   ```

4. This will open the graphical user interface of the application.

0.5 Navigating the Interface

The application has various tabs for data entry, querying, and settings. Each tab is dedicated to a specific function.

0.5.1 Data Entry

You can enter data for departments, faculty, programs, courses, etc. To enter data:

1. Click on the 'Data Entry' tab.
2. Choose the category (e.g., Departments, Faculty) you want to add data to.
3. Fill in the required fields.
4. Click 'Submit' to save the data.

0.5.2 Data Query

To query data:

1. Click on the 'Data Query' tab.
2. Choose the category you want to query (e.g., Department, Program).
3. Fill in the query parameters.
4. Click 'Submit' to view the results.

0.5.3 Settings

To reset the database:

1. Click on the 'Settings' menu in the top bar.
2. Select 'Reset Database'.
3. Confirm the action if prompted.

Implementation Manual

gui.py

1. **Validation Functions**: A collection of functions for validating user input in GUI forms. Each function corresponds to a specific type of input, ensuring correctness and consistency of data.
2. **Data Handling Functions**: These functions interact with the database to perform operations like adding, updating, and querying data based on user inputs from the GUI.
3. **GUI Setup Functions**: Responsible for initializing and configuring various components of the GUI, such as tabs, buttons, entry fields, etc.
4. **Entry and Query Field Setup**: Functions that create and manage entry fields for data input and query operations in the GUI.
5. **Main GUI Initialization**: The entry point for initializing the entire GUI application.

db.py

1. **SessionManager Class**: Manages database sessions, providing an interface for executing various database operations.
2. **Database Operation Functions**: Functions for adding and querying different entities in the database such as departments, faculty, programs, courses, etc.
3. **Specialized Query Functions**: Includes complex queries for retrieving specific data, like program evaluation results based on semester or year.

main.py

1. **Database Initialization**: Establishes the initial connection to the database and sets up the database schema.
2. **GUI Launch**: The starting point of the application, responsible for launching the GUI.

Program Code

main.py

```python
from sqlalchemy import event
from sqlalchemy.engine import Engine
from db import SessionManager
from gui import reset_database, initialize_db, initialize_gui


# @event.listens_for(Engine, "connect")
# def set_sqlite_pragma(dbapi_connection, connection_record):
#     cursor = dbapi_connection.cursor()
#     cursor.execute("PRAGMA foreign_keys=ON")
#     cursor.close()


if __name__ == "__main__":
    sqlite_url = "sqlite:///university_evaluation.db"
    username = "cs5330"
    password = "cs5330"
    hostname = "localhost"
    db_name = "dbprog"
    port = 3306
    mysql_connection_string = (
        f"mysql+mysqlconnector://{username}:{password}@{hostname}:{port}/{db_name}"
    )
    db_manager = SessionManager(mysql_connection_string)
    reset_database(db_manager.engine)
    initialize_db(db_manager)
    initialize_gui()
```

gui.py

```python
import tkinter as tk
from tkinter import ttk, messagebox
from sqlalchemy import create_engine
from db import Base
from random import random
import copy

DB = None


# Validation functions
def validate_non_empty(entry):
    return entry.get().strip() != ""


def validate_department_code(entry):
    return len(entry.get().strip()) <= 4 and entry.get().strip().isalnum()


def validate_email(entry):
    return "@" in entry.get().strip() and "." in entry.get().strip()


def validate_course_id(entry):
    return entry.get().strip() != ""


def validate_section_id(entry):
    return entry.get().strip().isdigit() and len(entry.get().strip()) <= 3


def validate_department_id(entry):
    return entry.get().strip().isdigit()


def validate_person_in_charge_id(entry):
    return entry.get().strip().isdigit()


def validate_enrollment_count(entry):
    return entry.get().strip().isdigit()


def validate_department_name(entry):
    return entry.get().strip() != ""


def validate_program_name(entry):
    return entry.get().strip() != ""


def validate_year(entry):
    return entry.get().strip() != "" and len(entry.get().strip(
    )) == 5 and entry.get().strip()[0:2].isdigit() and entry.get().strip(
    )[3:5].isdigit() and entry.get().strip()[2] == "-"


validation_functions = {
    "Code": validate_department_code,
    "Email": validate_email,
    "Course ID": validate_course_id,  # Assuming this is for course ID
    "Department ID": validate_department_id,
    "Person in Charge ID": validate_person_in_charge_id,
    "Enrollment Count": validate_enrollment_count,
    "Department Name": validate_department_name,
    "Program Name": validate_program_name,
```

```
    "Year": validate_year,
}


def set_entry_validity(entry, is_valid):
    if is_valid:
        entry.config(foreground="black")
    else:
        entry.config(foreground="red")


def check_entries(entries, submit_button):
    all_valid = True
    for field, entry in entries.items():
        is_valid = validate_non_empty(entry) and validation_functions.get(
            field, lambda e: True)(entry)
        set_entry_validity(entry, is_valid)
        all_valid &= is_valid
    submit_button.config(state="normal" if all_valid else "disabled")


def handle_data_submission(entries, status_label, category):
    is_valid = all(validate_non_empty(entry) for entry in entries.values())
    if is_valid:
        data = {field: entry.get() for field, entry in entries.items()}
        try:
            if category == "Departments":
                DB.add_department(data["Name"], data["Code"])
            elif category == "Faculty":
                DB.add_faculty(data["Name"], data["Email"], data["Rank"],
                               data["Code"])
            elif category == "Programs":
                DB.add_program(data["Name"], data["Code"],
                               data["Person in Charge ID"])
            elif category == "Courses":
                DB.add_course(data["ID"], data["Title"],
                              data["Description"], data["Code"])
            # Add similar branches for other categories
            status_label.config(
                text=f"Data for {category} successfully submitted.",
                fg="green")
        except Exception as e:
            status_label.config(text=str(e), fg="red")
    else:
        status_label.config(text="Invalid data in some fields.", fg="red")


def add_data_fields(tab, field_names, validation_functions, status_label,
                    category):
    entries = {}
    for field in field_names:
        frame = tk.Frame(tab)
        frame.pack(side="top", fill="x", padx=5, pady=5)

        label = tk.Label(frame, text=field, width=20)
        label.pack(side="left")

        entry = tk.Entry(frame)
        entry.pack(side="right", expand=True, fill="x")
        entry.bind(
            "<KeyRelease>",
            lambda event, e=entry: check_entries(entries, submit_button))
        entries[field] = entry

    submit_button = tk.Button(
        tab,
        text="Submit",
        state="disabled",
```

```
        command=lambda: handle_data_submission(entries, status_label, category
                                               ),
    )
    submit_button.pack(pady=10)

    check_entries(entries, submit_button)

    return entries


def handle_query_submission(entries, status_label, category):
    is_valid = all(validate_non_empty(entry) for entry in entries.values())
    if is_valid:
        data = {field: entry.get() for field, entry in entries.items()}
        try:
            results = ""

            if category == "Department":
                if data["Choice"] == "faculty":
                    results = DB.get_department_faculty_by_name(
                        data["Department Name"])
                elif data["Choice"] == "program":
                    results = DB.get_department_programs_by_name(
                        data["Department Name"])
            elif category == "Program":
                if data["Choice"] == "courses":
                    results = DB.get_program_courses_by_name(
                        data["Program Name"])
                elif data["Choice"] == "objectives":
                    results = DB.get_program_objectives_by_name(
                        data["Program Name"])
            elif category == "Semester Program":
                if data["Choice"] == "evaluation":
                    results = DB.get_results_by_semester(
                        data["Semester"], data["Program Name"])
            elif category == "Year":
                if data["Choice"] == "evaluation":
                    results = DB.get_results_by_year(data["Year"])

            final = '\n'.join([str(result) for result in results])
            status_label.config(
                text=f"Query for {category} successfully submitted.\n{final}",
                fg="green")
        except Exception as e:
            status_label.config(text=str(e), fg="red")
    else:
        status_label.config(text="Invalid data in some fields.", fg="red")


def add_query_fields(tab, field_names, validation_functions, status_label,
                     category):
    entries = {}
    for field in field_names:
        frame = tk.Frame(tab)
        frame.pack(side="top", fill="x", padx=5, pady=5)

        label = tk.Label(frame, text=field, width=20)
        label.pack(side="left")

        if field == "Choice":
            choice_var = tk.StringVar()
            choice_options = ["default"]
            if category == "Department":
                choice_options = ["faculty", "program"]
            elif category == "Program":
                choice_options = ["courses", "objectives"]
            elif category == "Semester Program":
                choice_options = ["evaluation"]
```

```
            elif category == "Year":
                choice_options = ["evaluation"]
            choice_dropdown = ttk.Combobox(frame,
                                           textvariable=choice_var,
                                           values=choice_options,
                                           state="readonly")
            choice_dropdown.set(choice_options[0])
            choice_dropdown.pack(side="right", expand=True, fill="x")
            entries[field] = choice_dropdown
        else:
            entry = tk.Entry(frame)
            entry.pack(side="right", expand=True, fill="x")
            entry.bind(
                "<KeyRelease>",
                lambda event, e=entry: check_entries(entries, submit_button))
            entries[field] = entry

    submit_button = tk.Button(
        tab,
        text="Submit",
        state="disabled",
        command=lambda: handle_query_submission(entries, status_label, category
                                                ),
    )
    submit_button.pack(pady=10)

    check_entries(entries, submit_button)

    return entries


def add_faculty_fields(tab, field_names, validation_functions, status_label):
    entries = {}
    for field in field_names:
        frame = tk.Frame(tab)
        frame.pack(side="top", fill="x", padx=5, pady=5)

        label = tk.Label(frame, text=field, width=20)
        label.pack(side="left")

        if field == "Rank":
            rank_var = tk.StringVar()
            rank_options = ["full", "associate", "assistant", "adjunct"]
            rank_dropdown = ttk.Combobox(frame,
                                         textvariable=rank_var,
                                         values=rank_options,
                                         state="readonly")
            rank_dropdown.set(rank_options[0])
            rank_dropdown.pack(side="right", expand=True, fill="x")
            entries[field] = rank_dropdown
        else:
            entry = tk.Entry(frame)
            entry.pack(side="right", expand=True, fill="x")
            entry.bind(
                "<KeyRelease>",
                lambda event, e=entry: check_entries(entries, submit_button))
            entries[field] = entry

    submit_button = tk.Button(tab,
                              text="Submit",
                              state="disabled",
                              command=lambda: handle_data_submission(
                                  entries, status_label, "Faculty"))
    submit_button.pack(pady=10)

    check_entries(entries, submit_button)

    return entries
```

```
271
272
273 def handle_course_program_assignment(entries, status_label):
274     is_valid = all(validate_non_empty(entry) for entry in entries.values())
275     if is_valid:
276         data = {field: entry.get() for field, entry in entries.items()}
277         try:
278             DB.assign_course_to_program(int(data["Program ID"]), data["Course ID"])
279             status_label.config(
280                 text="Course successfully assigned to program.", fg="green")
281         except Exception as e:
282             status_label.config(text=str(e), fg="red")
283     else:
284         status_label.config(text="Invalid data in some fields.", fg="red")
285
286
287 def add_course_program_assignment_fields(tab, validation_functions,
288                                          status_label):
289     entries = {}
290     fields = ["Program ID", "Course ID"]
291     for field in fields:
292         frame = tk.Frame(tab)
293         frame.pack(side="top", fill="x", padx=5, pady=5)
294         label = tk.Label(frame, text=field, width=20)
295         label.pack(side="left")
296         entry = tk.Entry(frame)
297         entry.pack(side="right", expand=True, fill="x")
298         entry.bind(
299             "<KeyRelease>",
300             lambda event, e=entry: check_entries(entries, submit_button))
301         entries[field] = entry
302
303     submit_button = tk.Button(tab,
304                               text="Assign",
305                               state="disabled",
306                               command=lambda: handle_course_program_assignment(
307                                   entries, status_label))
308     submit_button.pack(pady=10)
309     check_entries(entries, submit_button)
310
311
312 def handle_objective_assignment(entries, status_label):
313     is_valid = all(validate_non_empty(entry) for entry in entries.values())
314     if is_valid:
315         data = {field: entry.get() for field, entry in entries.items()}
316         try:
317             DB.assign_objective_to_course(data["Course ID"],
318                                           data["Objective ID"],
319                                           data["Program ID"])
320             status_label.config(text="Objective successfully assigned.",
321                                 fg="green")
322         except Exception as e:
323             status_label.config(text=str(e), fg="red")
324     else:
325         status_label.config(text="Invalid data in some fields.", fg="red")
326
327
328 def add_objective_assignment_fields(tab, validation_functions, status_label):
329     entries = {}
330     fields = ["Course ID", "Objective ID", "Program ID"]
331     for field in fields:
332         frame = tk.Frame(tab)
333         frame.pack(side="top", fill="x", padx=5, pady=5)
334         label = tk.Label(frame, text=field, width=20)
335         label.pack(side="left")
336         entry = tk.Entry(frame)
337         entry.pack(side="right", expand=True, fill="x")
338         entry.bind(
```

```
            "<KeyRelease>",
            lambda event, e=entry: check_entries(entries, submit_button))
        entries[field] = entry

    submit_button = tk.Button(
        tab,
        text="Assign",
        state="disabled",
        command=lambda: handle_objective_assignment(entries, status_label))
    submit_button.pack(pady=10)
    check_entries(entries, submit_button)


def handle_evaluation_submission(entries, status_label):
    is_valid = all(validate_non_empty(entry) for entry in entries.values())
    if is_valid:
        data = {field: entry.get() for field, entry in entries.items()}
        try:
            DB.add_section_evaluation(data["Section ID"], data["Objective ID"],
                                      data["Evaluation Method"],
                                      int(data["Students Met"]))
            status_label.config(
                text="Evaluation result successfully submitted.", fg="green")
        except Exception as e:
            status_label.config(text=str(e), fg="red")
    else:
        status_label.config(text="Invalid data in some fields.", fg="red")


def add_evaluation_fields(tab, validation_functions, status_label):
    entries = {}
    fields = [
        "Section ID", "Objective ID", "Evaluation Method", "Students Met"
    ]
    for field in fields:
        frame = tk.Frame(tab)
        frame.pack(side="top", fill="x", padx=5, pady=5)
        label = tk.Label(frame, text=field, width=20)
        label.pack(side="left")
        entry = tk.Entry(frame)
        entry.pack(side="right", expand=True, fill="x")
        entry.bind(
            "<KeyRelease>",
            lambda event, e=entry: check_entries(entries, submit_button))
        entries[field] = entry

    submit_button = tk.Button(
        tab,
        text="Submit",
        state="disabled",
        command=lambda: handle_evaluation_submission(entries, status_label))
    submit_button.pack(pady=10)
    check_entries(entries, submit_button)


def setup_data_entry_tab(notebook, status_label):
    data_entry_tab = ttk.Frame(notebook)
    notebook.add(data_entry_tab, text="Data Entry")

    data_entry_notebook = ttk.Notebook(data_entry_tab)
    data_entry_notebook.pack(expand=True, fill="both", padx=10, pady=10)

    departments_tab = ttk.Frame(data_entry_notebook)
    data_entry_notebook.add(departments_tab, text="Departments")
    department_fields = ["Name", "Code"]
    add_data_fields(departments_tab, department_fields,
                    {"Code": validate_department_code}, status_label,
                    "Departments")
```

```
407
408     faculty_tab = ttk.Frame(data_entry_notebook)
409     data_entry_notebook.add(faculty_tab, text="Faculty")
410     faculty_fields = ["Name", "Email", "Rank", "Code"]
411     add_faculty_fields(faculty_tab, faculty_fields, {"Email": validate_email, "Code": validate_department_code },
412                        status_label)
413
414     programs_tab = ttk.Frame(data_entry_notebook)
415     data_entry_notebook.add(programs_tab, text="Programs")
416     program_fields = ["Name", "Code", "Person in Charge ID"]
417     add_data_fields(programs_tab, program_fields, {}, status_label, "Programs")
418
419     courses_tab = ttk.Frame(data_entry_notebook)
420     data_entry_notebook.add(courses_tab, text="Courses")
421     course_fields = ["ID", "Title", "Description", "Code"]
422     add_data_fields(courses_tab, course_fields, {"ID": validate_course_id, "Code": validate_department_code },
423                     status_label, "Courses")
424
425     sections_tab = ttk.Frame(data_entry_notebook)
426     data_entry_notebook.add(sections_tab, text="Sections")
427     section_fields = [
428         "ID", "Course ID", "Semester", "Instructor ID", "Enrollment Count"
429     ]
430     add_data_fields(sections_tab, section_fields, {"ID": validate_section_id},
431                     status_label, "Sections")
432
433     objectives_tab = ttk.Frame(data_entry_notebook)
434     data_entry_notebook.add(objectives_tab, text="Learning Objectives")
435     objective_fields = ["Code", "Description", "Parent Objective Code"]
436     add_data_fields(objectives_tab, objective_fields, {}, status_label,
437                     "Learning Objectives")
438
439     course_program_tab = ttk.Frame(data_entry_notebook)
440     data_entry_notebook.add(course_program_tab,
441                             text="Assign Course to Program")
442     add_course_program_assignment_fields(course_program_tab,
443                                          validation_functions, status_label)
444
445     objective_assignment_tab = ttk.Frame(data_entry_notebook)
446     data_entry_notebook.add(objective_assignment_tab, text="Assign Objectives")
447     add_objective_assignment_fields(objective_assignment_tab,
448                                     validation_functions, status_label)
449
450     evaluation_tab = ttk.Frame(data_entry_notebook)
451     data_entry_notebook.add(evaluation_tab, text="Section Evaluation")
452     add_evaluation_fields(evaluation_tab, validation_functions, status_label)
453
454
455 def setup_data_query_tab(notebook, status_label):
456     data_query_tab = ttk.Frame(notebook)
457     notebook.add(data_query_tab, text="Data Query")
458
459     data_query_notebook = ttk.Notebook(data_query_tab)
460     data_query_notebook.pack(expand=True, fill="both", padx=10, pady=10)
461
462     # Department Queries
463     department_tab = ttk.Frame(data_query_notebook)
464     data_query_notebook.add(department_tab, text="Department")
465     department_fields = ["Choice", "Department Name"]
466     add_query_fields(department_tab, department_fields,
467                      {"Department Name": validate_department_name},
468                      status_label, "Department")
469
470     # Program Queries
471     program_tab = ttk.Frame(data_query_notebook)
472     data_query_notebook.add(program_tab, text="Program")
473     program_fields = ["Choice", "Program Name"]
474     add_query_fields(program_tab, program_fields,
```

```
                    {"Program Name": validate_program_name}, status_label,
                    "Program")

    # Semester + Program Queries
    semester_program_tab = ttk.Frame(data_query_notebook)
    data_query_notebook.add(semester_program_tab, text="Semester Program")
    semester_program_fields = ["Choice", "Semester", "Program Name"]
    add_query_fields(semester_program_tab, semester_program_fields,
                    {"Program Name": validate_program_name}, status_label,
                    "Semester Program")

    # Year Queries
    year_tab = ttk.Frame(data_query_notebook)
    data_query_notebook.add(year_tab, text="Year")
    year_fields = ["Choice", "Year"]
    add_query_fields(year_tab, year_fields, {"Year": validate_year},
                    status_label, "Year")


def reset_database(engine):
    Base.metadata.drop_all(engine)
    Base.metadata.create_all(engine)

def initialize_gui():
    window = tk.Tk()
    window.title("University Program Evaluation System")

    menubar = tk.Menu(window)
    window.config(menu=menubar)

    settings_menu = tk.Menu(menubar, tearoff=0)
    menubar.add_cascade(label="Settings", menu=settings_menu)

    settings_menu.add_command(label="Reset Database", command=reset_database)

    # status Label for showing messages
    status_label = tk.Label(window, text="", font="Helvetica 12", fg="red")
    status_label.pack(pady=(5, 10))

    main_notebook = ttk.Notebook(window)
    main_notebook.pack(expand=True, fill="both", padx=10, pady=10)

    # Setup Data Entry Tab
    setup_data_entry_tab(main_notebook, status_label)

    # Setup Data Query Tab
    setup_data_query_tab(main_notebook, status_label)

    window.mainloop()


def initialize_db(db):
    global DB
    DB = db

    # Department
    db.add_department("Cox School of Business", "BIZ")
    db.add_department("Lyle School of Engineering", "ENG")

    # Faculty
    db.add_faculty("Vishal Ahuja", "vishal.ahuja@smu.edu", "associate", "BIZ")
    db.add_faculty("Amy Altizer", "amy.altizer@smu.edu", "adjunct", "BIZ")
    db.add_faculty("Thomas Barry", "thomas.barry@smu.edu", "full", "BIZ")
    db.add_faculty("Wendy Bradley", "wendy.bradley@smu.edu", "assistant", "BIZ")

    db.add_faculty("Frank Coyle", "frank.coyle@smu.edu", "associate", "ENG")
    db.add_faculty("Qiguo Jing", "qiguo.jing@smu.edu", "adjunct", "ENG")
    db.add_faculty("Theodore Manikas", "theodore.manikas@smu.edu", "full", "ENG")
```

```
543     db.add_faculty("Corey Clark", "corey.clark@smu.edu", "assistant", "ENG")
544
545     # Program
546     db.add_program("Finance", "BIZ", 3) # 1
547     db.add_program("Accounting", "BIZ", 3) # 2
548     db.add_program("Marketing", "BIZ", 3) # 3
549
550     db.add_program("Computer Science", "ENG", 7) # 4
551     db.add_program("Computer Engineering", "ENG", 7) # 5
552     db.add_program("Creative Computing", "ENG", 7) # 6
553
554     # Course
555     dummy_text = ["Writing Proficiency", "Subject Knowledge", "Communication"]
556
557     db.add_course("BIZ1000", "Intro to Finance",
558                   "Introduction to all things Finance", "BIZ")
559     db.assign_course_to_program(1, "BIZ1000") # program_id, course_id
560     for i in range(1, 3):
561         isSubObjective = True if i % 2 == 0 else False
562         if isSubObjective:
563             options = [j for j in range(1, i)]
564             # id, description, parent_id (optional)
565             db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
566             # course_id, objective_id, program_id
567             db.assign_objective_to_course("BIZ1000", i, 1)
568         else:
569             db.add_learning_objective(i, dummy_text[int(random() * 3)])
570     db.add_course("BIZ1100", "Intro to Marketing",
571                   "Introduction to all things Marketing", "BIZ")
572     db.assign_course_to_program(3, "BIZ1100")
573     for i in range(3, 5):
574         isSubObjective = True if i % 2 == 0 else False
575         if isSubObjective:
576             options = [j for j in range(3, i)]
577             # id, description, parent_id (optional)
578             db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
579             # course_id, objective_id, program_id
580             db.assign_objective_to_course("BIZ1100", i, 3)
581         else:
582             db.add_learning_objective(i, dummy_text[int(random() * 3)])
583     db.add_course("BIZ1200", "Intro to Accounting",
584                   "Introduction to all things Accounting", "BIZ")
585     db.assign_course_to_program(2, "BIZ1200")
586     for i in range(5, 7):
587         isSubObjective = True if i % 2 == 0 else False
588         if isSubObjective:
589             options = [j for j in range(5, i + 1)]
590             # id, description, parent_id (optional)
591             db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
592             # course_id, objective_id, program_id
593             db.assign_objective_to_course("BIZ1200", i, 2)
594         else:
595             db.add_learning_objective(i, dummy_text[int(random() * 3)])
596     db.add_course("BIZ2000", "Intermediate Finance",
597                   "Intermediate class for all things Finance", "BIZ")
598     db.assign_course_to_program(1, "BIZ2000")
599     for i in range(7, 9):
600         isSubObjective = True if i % 2 == 0 else False
601         if isSubObjective:
602             options = [j for j in range(7, i)]
603             # id, description, parent_id (optional)
604             db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
605             # course_id, objective_id, program_id
606             db.assign_objective_to_course("BIZ2000", i, 1)
607         else:
608             db.add_learning_objective(i, dummy_text[int(random() * 3)])
609     db.add_course("BIZ2100", "Intermediate Marketing",
610                   "Intermediate class for all things Marketing", "BIZ")
```

```
    db.assign_course_to_program(3, "BIZ2100")
    for i in range(9, 11):
        isSubObjective = True if i % 2 == 0 else False
        if isSubObjective:
            options = [j for j in range(9, i)]
            # id, description, parent_id (optional)
            db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
            # course_id, objective_id, program_id
            db.assign_objective_to_course("BIZ2100", i, 3)
        else:
            db.add_learning_objective(i, dummy_text[int(random() * 3)])
    db.add_course("BIZ2200", "Intermediate Accounting",
                  "Intermediate class for all things Accounting", "BIZ")
    db.assign_course_to_program(2, "BIZ2200")
    for i in range(11, 13):
        isSubObjective = True if i % 2 == 0 else False
        if isSubObjective:
            options = [j for j in range(11, i)]
            # id, description, parent_id (optional)
            db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
            # course_id, objective_id, program_id
            db.assign_objective_to_course("BIZ2200", i, 2)
        else:
            db.add_learning_objective(i, dummy_text[int(random() * 3)])
    db.add_course("BIZ3000", "Advanced Finance",
                  "Advanced class for all things Finance", "BIZ")
    db.assign_course_to_program(1, "BIZ3000")
    for i in range(13, 15):
        isSubObjective = True if i % 2 == 0 else False
        if isSubObjective:
            options = [j for j in range(13, i)]
            # id, description, parent_id (optional)
            db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
            # course_id, objective_id, program_id
            db.assign_objective_to_course("BIZ3000", i, 1)
        else:
            db.add_learning_objective(i, dummy_text[int(random() * 3)])
    db.add_course("BIZ3100", "Advanced Marketing",
                  "Advanced class for all things Marketing", "BIZ")
    db.assign_course_to_program(3, "BIZ3100")
    for i in range(15, 17):
        isSubObjective = True if i % 2 == 0 else False
        if isSubObjective:
            options = [j for j in range(15, i)]
            # id, description, parent_id (optional)
            db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
            # course_id, objective_id, program_id
            db.assign_objective_to_course("BIZ3100", i, 3)
        else:
            db.add_learning_objective(i, dummy_text[int(random() * 3)])
    db.add_course("BIZ3200", "Advanced Accounting",
                  "Advanced class for all things Accounting", "BIZ")
    db.assign_course_to_program(2, "BIZ3200")
    for i in range(17, 19):
        isSubObjective = True if i % 2 == 0 else False
        if isSubObjective:
            options = [j for j in range(17, i)]
            # id, description, parent_id (optional)
            db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
            # course_id, objective_id, program_id
            db.assign_objective_to_course("BIZ3200", i, 2)
        else:
            db.add_learning_objective(i, dummy_text[int(random() * 3)])

    db.add_course("ENG1000", "Intro to Computer Science",
                  "Introduction to all things Computer Science", "ENG")
    db.assign_course_to_program(4, "ENG1000")
    for i in range(19, 21):
```

```
679             isSubObjective = True if i % 2 == 0 else False
680             if isSubObjective:
681                 options = [j for j in range(19, i)]
682                 # id, description, parent_id (optional)
683                 db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
684                 # course_id, objective_id, program_id
685                 db.assign_objective_to_course("ENG1000", i, 4)
686             else:
687                 db.add_learning_objective(i, dummy_text[int(random() * 3)])
688         db.add_course("ENG1100", "Intro to Computer Engineering",
689                       "Introduction to all things Computer Engineering", "ENG")
690         db.assign_course_to_program(5, "ENG1100")
691         for i in range(21, 23):
692             isSubObjective = True if i % 2 == 0 else False
693             if isSubObjective:
694                 options = [j for j in range(21, i)]
695                 # id, description, parent_id (optional)
696                 db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
697                 # course_id, objective_id, program_id
698                 db.assign_objective_to_course("ENG1100", i, 5)
699             else:
700                 db.add_learning_objective(i, dummy_text[int(random() * 3)])
701         db.add_course("ENG1200", "Intro to Creative Computing",
702                       "Introduction to all things Creative Computing", "ENG")
703         db.assign_course_to_program(6, "ENG1200")
704         for i in range(23, 25):
705             isSubObjective = True if i % 2 == 0 else False
706             if isSubObjective:
707                 options = [j for j in range(23, i)]
708                 # id, description, parent_id (optional)
709                 db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
710                 # course_id, objective_id, program_id
711                 db.assign_objective_to_course("ENG1200", i, 6)
712             else:
713                 db.add_learning_objective(i, dummy_text[int(random() * 3)])
714         db.add_course("ENG2000", "Intermediate Computer Science",
715                       "Intermediate class for all things Computer Science", "ENG")
716         db.assign_course_to_program(4, "ENG2000")
717         for i in range(25, 27):
718             isSubObjective = True if i % 2 == 0 else False
719             if isSubObjective:
720                 options = [j for j in range(25, i)]
721                 # id, description, parent_id (optional)
722                 db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
723                 # course_id, objective_id, program_id
724                 db.assign_objective_to_course("ENG2000", i, 4)
725             else:
726                 db.add_learning_objective(i, dummy_text[int(random() * 3)])
727         db.add_course("ENG2100", "Intermediate Computer Engineering",
728                       "Intermediate class for all things Computer Engineering", "ENG")
729         db.assign_course_to_program(5, "ENG2100")
730         for i in range(27, 29):
731             isSubObjective = True if i % 2 == 0 else False
732             if isSubObjective:
733                 options = [j for j in range(27, i)]
734                 # id, description, parent_id (optional)
735                 db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
736                 # course_id, objective_id, program_id
737                 db.assign_objective_to_course("ENG2100", i, 5)
738             else:
739                 db.add_learning_objective(i, dummy_text[int(random() * 3)])
740         db.add_course("ENG2200", "Intermediate Creative Computing",
741                       "Intermediate class for all things Creative Computing", "ENG")
742         db.assign_course_to_program(6, "ENG2200")
743         for i in range(29, 31):
744             isSubObjective = True if i % 2 == 0 else False
745             if isSubObjective:
746                 options = [j for j in range(29, i)]
```

```
            # id, description, parent_id (optional)
            db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
            # course_id, objective_id, program_id
            db.assign_objective_to_course("ENG2200", i, 6)
        else:
            db.add_learning_objective(i, dummy_text[int(random() * 3)])
    db.add_course("ENG3000", "Advanced Computer Science",
                  "Advanced class for all things Computer Science", "ENG")
    db.assign_course_to_program(4, "ENG3000")
    for i in range(31, 33):
        isSubObjective = True if i % 2 == 0 else False
        if isSubObjective:
            options = [j for j in range(31, i)]
            # id, description, parent_id (optional)
            db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
            # course_id, objective_id, program_id
            db.assign_objective_to_course("ENG3000", i, 4)
        else:
            db.add_learning_objective(i, dummy_text[int(random() * 3)])
    db.add_course("ENG3100", "Advanced Computer Engineering",
                  "Advanced class for all things Computer Engineering", "ENG")
    db.assign_course_to_program(5, "ENG3100")
    for i in range(33, 35):
        isSubObjective = True if i % 2 == 0 else False
        if isSubObjective:
            options = [j for j in range(33, i)]
            # id, description, parent_id (optional)
            db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
            # course_id, objective_id, program_id
            db.assign_objective_to_course("ENG3100", i, 5)
        else:
            db.add_learning_objective(i, dummy_text[int(random() * 3)])
    db.add_course("ENG3200", "Advanced Creative Computing",
                  "Advanced class for all things Creative Computing", "ENG")
    db.assign_course_to_program(6, "ENG3200")
    for i in range(35, 37):
        isSubObjective = True if i % 2 == 0 else False
        if isSubObjective:
            options = [j for j in range(35, i)]
            # id, description, parent_id (optional)
            db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
            # course_id, objective_id, program_id
            db.assign_objective_to_course("ENG3200", i, 6)
        else:
            db.add_learning_objective(i, dummy_text[int(random() * 3)])

    # Section
    semester = ["Fall", "Spring", "Summer"]
    year = [21, 22, 23, 24, 25]
    eval = ["Exam", "Homework", "Participation"]
    # db.add_section(200, "Fall", 23, "BIZ1000", 3, 40) # number, semester, year, course_id, instructor_id, enrollment_count
    # db.add_section_evaluation(1, 1, "Good", 20) # section_id, objective_id, evaluation_method, students_met
    secid = 0
    for i in range(0, 51, 10):
        secnum = int(random() * 10) + i
        sem = semester[int(random() * 3)]
        yr = year[int(random() * 5)]
        cid = "BIZ1000"
        iid = int(random() * 4) + 1
        numstudents = int(random() * 50)
        db.add_section(secnum, sem, yr, cid, iid, numstudents)
        secid += 1
        for j in range(1, 3):
            ev = eval[int(random() * len(eval))]
            numEvStudents = int(random() * numstudents)
            db.add_section_evaluation(secid, j, ev, numEvStudents)
    for i in range(0, 51, 10):
        secnum=int(random() * 10) + i
```

```
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "BIZ1100"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(3, 5):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "BIZ1200"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(5, 7):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "BIZ2000"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(7, 9):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "BIZ2100"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(9, 11):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "BIZ2200"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(11, 13):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "BIZ3000"
```

```
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(13, 15):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "BIZ3100"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(15, 17):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "BIZ3200"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(17, 19):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "ENG1000"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(19, 21):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "ENG1100"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(21, 23):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "ENG1200"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
```

```
            secid += 1
            for j in range(23, 25):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "ENG2000"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(25, 27):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "ENG2100"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(27, 29):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "ENG2200"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(29, 31):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "ENG3000"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(31, 33):
                ev = eval[int(random() * len(eval))]
                numEvStudents = int(random() * numstudents)
                db.add_section_evaluation(secid, j, ev, numEvStudents)
        for i in range(0, 51, 10):
            secnum = int(random() * 10) + i
            sem = semester[int(random() * 3)]
            yr = year[int(random() * 5)]
            cid = "ENG3100"
            iid = int(random() * 4) + 1
            numstudents = int(random() * 50)
            db.add_section(secnum, sem, yr, cid, iid, numstudents)
            secid += 1
            for j in range(33, 35):
                ev = eval[int(random() * len(eval))]
```

```
            numEvStudents = int(random() * numstudents)
            db.add_section_evaluation(secid, j, ev, numEvStudents)
    for i in range(0, 51, 10):
        secnum = int(random() * 10) + i
        sem = semester[int(random() * 3)]
        yr = year[int(random() * 5)]
        cid = "ENG3200"
        iid = int(random() * 4) + 1
        numstudents = int(random() * 50)
        db.add_section(secnum, sem, yr, cid, iid, numstudents)
        secid += 1
        for j in range(35, 37):
            ev = eval[int(random() * len(eval))]
            numEvStudents = int(random() * numstudents)
            db.add_section_evaluation(secid, j, ev, numEvStudents)

    print('db initialized successfully')
```

db.py

```python
from sqlalchemy import (
    Column,
    Integer,
    String,
    ForeignKey,
    Text,
    create_engine,
    text,
)
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship, sessionmaker

Base = declarative_base()


class Department(Base):
    __tablename__ = "departments"
    id = Column(Integer, primary_key=True)
    name = Column(String(255), unique=True)
    code = Column(String(4), unique=True)
    faculty = relationship("Faculty", backref="department")
    programs = relationship("Program", backref="department")


class Faculty(Base):
    __tablename__ = "faculty"
    id = Column(Integer, primary_key=True)
    name = Column(String(255))
    email = Column(String(255), unique=True)
    rank = Column(String(50))
    department_code = Column(String(4), ForeignKey("departments.code"))


class Program(Base):
    __tablename__ = "programs"
    id = Column(Integer, primary_key=True)
    name = Column(String(255), unique=True)
    department_code = Column(String(4), ForeignKey("departments.code"))
    in_charge_id = Column(Integer, ForeignKey("faculty.id"))
    courses = relationship("Course", secondary="program_courses")


class Course(Base):
    __tablename__ = "courses"
    id = Column(String(10), primary_key=True)
    title = Column(String(255))
    description = Column(Text)
    department_code = Column(String(4), ForeignKey("departments.code"))
    sections = relationship("Section", backref="course")


class Section(Base):
    __tablename__ = "sections"
    id = Column(Integer, primary_key=True)
    number = Column(Integer)
    semester = Column(String(20))
    year = Column(Integer)
    course_id = Column(String(10), ForeignKey("courses.id"))
    instructor_id = Column(Integer, ForeignKey("faculty.id"))
    enrollment_count = Column(Integer)


class LearningObjective(Base):
    __tablename__ = "learning_objectives"
    id = Column(String(50), primary_key=True)
    description = Column(Text)
```

```
    parent_id = Column(String(50), ForeignKey("learning_objectives.id"))
    sub_objectives = relationship("LearningObjective")


class ProgramCourses(Base):
    __tablename__ = "program_courses"
    program_id = Column(Integer, ForeignKey("programs.id"), primary_key=True)
    course_id = Column(String(10), ForeignKey("courses.id"), primary_key=True)


class CourseObjectives(Base):
    __tablename__ = "course_objectives"
    course_id = Column(String(10), ForeignKey("courses.id"), primary_key=True)
    objective_id = Column(
        String(50), ForeignKey("learning_objectives.id"), primary_key=True
    )
    program_id = Column(Integer, ForeignKey("programs.id"), primary_key=True)


class SectionEvaluations(Base):
    __tablename__ = "section_evaluations"
    section_id = Column(Integer, ForeignKey("sections.id"), primary_key=True)
    objective_id = Column(String(50), ForeignKey("learning_objectives.id"), primary_key=True)
    evaluation_method = Column(String(255), primary_key=True)
    students_met = Column(Integer)


class SessionManager:
    def __init__(self, database_uri):
        self.engine = create_engine(database_uri)
        Session = sessionmaker(bind=self.engine)
        self.session = Session()

    def add(self, entry):
        try:
            self.session.add(entry)
            self.session.commit()
        except:
            self.session.rollback()
            raise

    def add_department(self, name, code):
        new_department = Department(name=name, code=code)
        self.add(new_department)

    def add_faculty(self, name, email, rank, department_code):
        new_faculty = Faculty(
            name=name, email=email, rank=rank, department_code=department_code
        )
        self.add(new_faculty)

    def add_program(self, name, department_code, in_charge_id):
        new_program = Program(
            name=name, department_code=department_code, in_charge_id=in_charge_id
        )
        self.add(new_program)

    def add_course(self, id, title, description, department_code):
        new_course = Course(
            id=id, title=title, description=description, department_code=department_code
        )
        self.add(new_course)

    def add_section(
        self, number, semester, year, course_id, instructor_id, enrollment_count
    ):
        new_section = Section(
            number=number,
```

```
            semester=semester,
            year=year,
            course_id=course_id,
            instructor_id=instructor_id,
            enrollment_count=enrollment_count,
        )
        self.add(new_section)

    def add_learning_objective(self, id, description, parent_id=None):
        new_objective = LearningObjective(
            id=id, description=description, parent_id=parent_id
        )
        self.add(new_objective)

    def assign_course_to_program(self, program_id, course_id):
        new_assignment = ProgramCourses(program_id=program_id, course_id=course_id)
        self.add(new_assignment)

    def assign_objective_to_course(self, course_id, objective_id, program_id):
        new_assignment = CourseObjectives(
            course_id=course_id, objective_id=objective_id, program_id=program_id
        )
        self.add(new_assignment)

    def add_section_evaluation(
        self, section_id, objective_id, evaluation_method, students_met
    ):
        new_evaluation = SectionEvaluations(
            section_id=section_id,
            objective_id=objective_id,
            evaluation_method=evaluation_method,
            students_met=students_met,
        )
        self.add(new_evaluation)

    def query(self, query):
        return [[attr for attr in row] for row in self.session.execute(text(query))]

    # List all of its programs
    def get_department_programs_by_name(self, department_name):
        return self.query(
            f"SELECT p.name "
            f"FROM departments AS d "
            f"JOIN programs AS p "
            f"ON d.code = p.department_code "
            f"WHERE d.name= '{department_name}'"
        )

    # List all of its faculty (including what program each faculty is in charge of, if there is one)
    def get_department_faculty_by_name(self, department_name):
        return self.query(
            f"SELECT f.name, p.name "
            f"FROM departments AS d "
            f"JOIN faculty AS f "
            f"ON d.code = f.department_code "
            f"LEFT JOIN programs AS p "
            f"ON f.id = p.in_charge_id "
            f"WHERE d.name = '{department_name}'"
        )

    # List all the courses, together with the objectives/sub-objectives association with year

    def get_program_courses_by_name(self, program_name):
        return self.query(
            f"SELECT c.title, lo.description "
            f"FROM courses AS c "
            f"JOIN program_courses AS pc "
            f"ON c.id = pc.course_id "
```

```
            f"JOIN programs AS p "
            f"ON p.id = pc.program_id "
            f"LEFT JOIN course_objectives AS co "
            f"ON co.course_id = c.id "
            f"LEFT JOIN learning_objectives AS lo "
            f"ON co.objective_id = lo.id "
            f"WHERE p.name = '{program_name}'"
        )

    # List all of the objectives
    def get_program_objectives_by_name(self, program_name):
        return self.query(
            f"SELECT DISTINCT lo.description "
            f"FROM courses AS c "
            f"JOIN program_courses AS pc "
            f"ON c.id = pc.course_id "
            f"JOIN programs AS p "
            f"ON p.id = pc.program_id "
            f"JOIN course_objectives AS co "
            f"ON co.course_id = c.id "
            f"JOIN learning_objectives AS lo "
            f"ON co.objective_id = lo.id "
            f"WHERE p.name = '{program_name}'"
        )

    # List all of the evaluation results for each objective/sub-objective (If data for some sections has not been entered, indica
    def get_results_by_semester(self, semester, program_name):
        semester = semester.split(" ")
        return self.query(
            f"SELECT c.title, s.number, se.evaluation_method, se.students_met "
            f"FROM courses AS c "
            f"JOIN sections AS s "
            f"ON c.id = s.course_id "
            f"JOIN program_courses AS pc "
            f"ON c.id = pc.course_id "
            f"JOIN programs AS p "
            f"ON p.id = pc.program_id "
            f"LEFT OUTER JOIN section_evaluations AS se "
            f"ON s.id = se.section_id "
            f"WHERE s.semester = '{semester[0]}' "  # semester[0] will contain "Fall" "Spring" etc
            f"AND s.year = {semester[1]} "  # semester[1] will contain year (i.e. 23, 24, etc.)
            f"AND p.name = '{program_name}'"
        )

    # List all of the evaluation results for each objective/sub-objective
    # Show course/section involved in evaluation, list result for each course/section, and aggregate the result to show the numbe
    def get_results_by_year(self, year):
        year = year.split("-")
        return self.query(
            f"SELECT lo.description, c.title, s.number, se.evaluation_method, se.students_met, ROUND(se.students_met * 100.0/s.e
            f"FROM courses AS c "
            f"JOIN sections AS s "
            f"ON c.id = s.course_id "
            f"JOIN course_objectives AS co "
            f"ON c.id = co.course_id "
            f"JOIN learning_objectives AS lo "
            f"ON co.objective_id = lo.id "
            f"JOIN section_evaluations AS se "
            f"ON s.id = se.section_id AND lo.id = se.objective_id "
            f"WHERE (s.year = {year[0]} "
            f"AND s.semester = 'Summer') "
            f"OR (s.year = {year[0]} "
            f"AND s.semester = 'Fall') "
            f"OR (s.year = {year[1]} "
            f"AND s.semester = 'Spring') "
            f"GROUP BY lo.id, c.id, s.id, se.evaluation_method "
            f"ORDER BY c.id, s.id"
        )
```