

## 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:

```
# 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

```
1  from sqlalchemy import event
2  from sqlalchemy.engine import Engine
3  from db import SessionManager
4  from gui import reset_database, initialize_db, initialize_gui
5
6
7  # @event.listens_for(Engine, "connect")
8  # def set_sqlite_pragma(dbapi_connection, connection_record):
9  #     cursor = dbapi_connection.cursor()
10 #     cursor.execute("PRAGMA foreign_keys=ON")
11 #     cursor.close()
12
13
14 if __name__ == "__main__":
15     sqlite_url = "sqlite:///university_evaluation.db"
16     username = "cs5330"
17     password = "cs5330"
18     hostname = "localhost"
19     db_name = "dbprog"
20     port = 3306
21     mysql_connection_string = (
22         f"mysql+mysqlconnector://{username}:{password}@{hostname}:{port}/{db_name}"
23     )
24     db_manager = SessionManager(mysql_connection_string)
25     reset_database(db_manager.engine)
26     initialize_db(db_manager)
27     initialize_gui()
```

**gui.py**

```

1  import tkinter as tk
2  from tkinter import ttk, messagebox
3  from sqlalchemy import create_engine
4  from db import Base
5  from random import random
6  import copy
7
8  DB = None
9
10
11  # Validation functions
12  def validate_non_empty(entry):
13      return entry.get().strip() != ""
14
15
16  def validate_department_code(entry):
17      return len(entry.get().strip()) <= 4 and entry.get().strip().isalnum()
18
19
20  def validate_email(entry):
21      return "@" in entry.get().strip() and "." in entry.get().strip()
22
23
24  def validate_course_id(entry):
25      return entry.get().strip() != ""
26
27
28  def validate_section_id(entry):
29      return entry.get().strip().isdigit() and len(entry.get().strip()) <= 3
30
31
32  def validate_department_id(entry):
33      return entry.get().strip().isdigit()
34
35
36  def validate_person_in_charge_id(entry):
37      return entry.get().strip().isdigit()
38
39
40  def validate_enrollment_count(entry):
41      return entry.get().strip().isdigit()
42
43
44  def validate_department_name(entry):
45      return entry.get().strip() != ""
46
47
48  def validate_program_name(entry):
49      return entry.get().strip() != ""
50
51
52  def validate_year(entry):
53      return entry.get().strip() != "" and len(entry.get().strip(
54      )) == 5 and entry.get().strip()[0:2].isdigit() and entry.get().strip(
55      )[3:5].isdigit() and entry.get().strip()[2] == "-"
56
57
58  validation_functions = {
59      "Code": validate_department_code,
60      "Email": validate_email,
61      "Course ID": validate_course_id, # Assuming this is for course ID
62      "Department ID": validate_department_id,
63      "Person in Charge ID": validate_person_in_charge_id,
64      "Enrollment Count": validate_enrollment_count,
65      "Department Name": validate_department_name,
66      "Program Name": validate_program_name,

```

```

67     "Year": validate_year,
68 }
69
70
71 def set_entry_validity(entry, is_valid):
72     if is_valid:
73         entry.config(foreground="black")
74     else:
75         entry.config(foreground="red")
76
77
78 def check_entries(entries, submit_button):
79     all_valid = True
80     for field, entry in entries.items():
81         is_valid = validate_non_empty(entry) and validation_functions.get(
82             field, lambda e: True)(entry)
83         set_entry_validity(entry, is_valid)
84         all_valid &= is_valid
85     submit_button.config(state="normal" if all_valid else "disabled")
86
87
88 def handle_data_submission(entries, status_label, category):
89     is_valid = all(validate_non_empty(entry) for entry in entries.values())
90     if is_valid:
91         data = {field: entry.get() for field, entry in entries.items()}
92         try:
93             if category == "Departments":
94                 DB.add_department(data["Name"], data["Code"])
95             elif category == "Faculty":
96                 DB.add_faculty(data["Name"], data["Email"], data["Rank"],
97                               data["Code"])
98             elif category == "Programs":
99                 DB.add_program(data["Name"], data["Code"],
100                                data["Person in Charge ID"])
101             elif category == "Courses":
102                 DB.add_course(data["ID"], data["Title"],
103                               data["Description"], data["Code"])
104             # Add similar branches for other categories
105             status_label.config(
106                 text=f"Data for {category} successfully submitted.",
107                 fg="green")
108         except Exception as e:
109             status_label.config(text=str(e), fg="red")
110     else:
111         status_label.config(text="Invalid data in some fields.", fg="red")
112
113
114 def add_data_fields(tab, field_names, validation_functions, status_label,
115                    category):
116     entries = {}
117     for field in field_names:
118         frame = tk.Frame(tab)
119         frame.pack(side="top", fill="x", padx=5, pady=5)
120
121         label = tk.Label(frame, text=field, width=20)
122         label.pack(side="left")
123
124         entry = tk.Entry(frame)
125         entry.pack(side="right", expand=True, fill="x")
126         entry.bind(
127             "<KeyRelease>",
128             lambda event, e=entry: check_entries(entries, submit_button))
129         entries[field] = entry
130
131     submit_button = tk.Button(
132         tab,
133         text="Submit",
134         state="disabled",

```

```

135         command=lambda: handle_data_submission(entries, status_label, category
136                                           ),
137     )
138     submit_button.pack(pady=10)
139
140     check_entries(entries, submit_button)
141
142     return entries
143
144
145 def handle_query_submission(entries, status_label, category):
146     is_valid = all(validate_non_empty(entry) for entry in entries.values())
147     if is_valid:
148         data = {field: entry.get() for field, entry in entries.items()}
149         try:
150             results = ""
151
152             if category == "Department":
153                 if data["Choice"] == "faculty":
154                     results = DB.get_department_faculty_by_name(
155                         data["Department Name"])
156                 elif data["Choice"] == "program":
157                     results = DB.get_department_programs_by_name(
158                         data["Department Name"])
159             elif category == "Program":
160                 if data["Choice"] == "courses":
161                     results = DB.get_program_courses_by_name(
162                         data["Program Name"])
163                 elif data["Choice"] == "objectives":
164                     results = DB.get_program_objectives_by_name(
165                         data["Program Name"])
166             elif category == "Semester Program":
167                 if data["Choice"] == "evaluation":
168                     results = DB.get_results_by_semester(
169                         data["Semester"], data["Program Name"])
170             elif category == "Year":
171                 if data["Choice"] == "evaluation":
172                     results = DB.get_results_by_year(data["Year"])
173
174             final = '\n'.join([str(result) for result in results])
175             status_label.config(
176                 text=f"Query for {category} successfully submitted.\n{final}",
177                 fg="green")
178         except Exception as e:
179             status_label.config(text=str(e), fg="red")
180     else:
181         status_label.config(text="Invalid data in some fields.", fg="red")
182
183
184 def add_query_fields(tab, field_names, validation_functions, status_label,
185                     category):
186     entries = {}
187     for field in field_names:
188         frame = tk.Frame(tab)
189         frame.pack(side="top", fill="x", padx=5, pady=5)
190
191         label = tk.Label(frame, text=field, width=20)
192         label.pack(side="left")
193
194         if field == "Choice":
195             choice_var = tk.StringVar()
196             choice_options = ["default"]
197             if category == "Department":
198                 choice_options = ["faculty", "program"]
199             elif category == "Program":
200                 choice_options = ["courses", "objectives"]
201             elif category == "Semester Program":
202                 choice_options = ["evaluation"]

```



```

203         elif category == "Year":
204             choice_options = ["evaluation"]
205             choice_dropdown = ttk.Combobox(frame,
206                                           textvariable=choice_var,
207                                           values=choice_options,
208                                           state="readonly")
209             choice_dropdown.set(choice_options[0])
210             choice_dropdown.pack(side="right", expand=True, fill="x")
211             entries[field] = choice_dropdown
212         else:
213             entry = tk.Entry(frame)
214             entry.pack(side="right", expand=True, fill="x")
215             entry.bind(
216                 "<KeyRelease>",
217                 lambda event, e=entry: check_entries(entries, submit_button))
218             entries[field] = entry
219
220     submit_button = tk.Button(
221         tab,
222         text="Submit",
223         state="disabled",
224         command=lambda: handle_query_submission(entries, status_label, category
225         ),
226     )
227     submit_button.pack(pady=10)
228
229     check_entries(entries, submit_button)
230
231     return entries
232
233
234 def add_faculty_fields(tab, field_names, validation_functions, status_label):
235     entries = {}
236     for field in field_names:
237         frame = tk.Frame(tab)
238         frame.pack(side="top", fill="x", padx=5, pady=5)
239
240         label = tk.Label(frame, text=field, width=20)
241         label.pack(side="left")
242
243         if field == "Rank":
244             rank_var = tk.StringVar()
245             rank_options = ["full", "associate", "assistant", "adjunct"]
246             rank_dropdown = ttk.Combobox(frame,
247                                         textvariable=rank_var,
248                                         values=rank_options,
249                                         state="readonly")
250             rank_dropdown.set(rank_options[0])
251             rank_dropdown.pack(side="right", expand=True, fill="x")
252             entries[field] = rank_dropdown
253         else:
254             entry = tk.Entry(frame)
255             entry.pack(side="right", expand=True, fill="x")
256             entry.bind(
257                 "<KeyRelease>",
258                 lambda event, e=entry: check_entries(entries, submit_button))
259             entries[field] = entry
260
261     submit_button = tk.Button(tab,
262                               text="Submit",
263                               state="disabled",
264                               command=lambda: handle_data_submission(
265                                   entries, status_label, "Faculty"))
266     submit_button.pack(pady=10)
267
268     check_entries(entries, submit_button)
269
270     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(

```

```

339         "<KeyRelease>",
340         lambda event, e=entry: check_entries(entries, submit_button))
341     entries[field] = entry
342
343     submit_button = tk.Button(
344         tab,
345         text="Assign",
346         state="disabled",
347         command=lambda: handle_objective_assignment(entries, status_label))
348     submit_button.pack(pady=10)
349     check_entries(entries, submit_button)
350
351
352 def handle_evaluation_submission(entries, status_label):
353     is_valid = all(validate_non_empty(entry) for entry in entries.values())
354     if is_valid:
355         data = {field: entry.get() for field, entry in entries.items()}
356         try:
357             DB.add_section_evaluation(data["Section ID"], data["Objective ID"],
358                                     data["Evaluation Method"],
359                                     int(data["Students Met"]))
360             status_label.config(
361                 text="Evaluation result successfully submitted.", fg="green")
362         except Exception as e:
363             status_label.config(text=str(e), fg="red")
364     else:
365         status_label.config(text="Invalid data in some fields.", fg="red")
366
367
368 def add_evaluation_fields(tab, validation_functions, status_label):
369     entries = {}
370     fields = [
371         "Section ID", "Objective ID", "Evaluation Method", "Students Met"
372     ]
373     for field in fields:
374         frame = tk.Frame(tab)
375         frame.pack(side="top", fill="x", padx=5, pady=5)
376         label = tk.Label(frame, text=field, width=20)
377         label.pack(side="left")
378         entry = tk.Entry(frame)
379         entry.pack(side="right", expand=True, fill="x")
380         entry.bind(
381             "<KeyRelease>",
382             lambda event, e=entry: check_entries(entries, submit_button))
383         entries[field] = entry
384
385     submit_button = tk.Button(
386         tab,
387         text="Submit",
388         state="disabled",
389         command=lambda: handle_evaluation_submission(entries, status_label))
390     submit_button.pack(pady=10)
391     check_entries(entries, submit_button)
392
393
394 def setup_data_entry_tab(notebook, status_label):
395     data_entry_tab = ttk.Frame(notebook)
396     notebook.add(data_entry_tab, text="Data Entry")
397
398     data_entry_notebook = ttk.Notebook(data_entry_tab)
399     data_entry_notebook.pack(expand=True, fill="both", padx=10, pady=10)
400
401     departments_tab = ttk.Frame(data_entry_notebook)
402     data_entry_notebook.add(departments_tab, text="Departments")
403     department_fields = ["Name", "Code"]
404     add_data_fields(departments_tab, department_fields,
405                    {"Code": validate_department_code}, status_label,
406                    "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,

```

```

475         {"Program Name": validate_program_name}, status_label,
476         "Program")
477
478     # Semester + Program Queries
479     semester_program_tab = ttk.Frame(data_query_notebook)
480     data_query_notebook.add(semester_program_tab, text="Semester Program")
481     semester_program_fields = ["Choice", "Semester", "Program Name"]
482     add_query_fields(semester_program_tab, semester_program_fields,
483                     {"Program Name": validate_program_name}, status_label,
484                     "Semester Program")
485
486     # Year Queries
487     year_tab = ttk.Frame(data_query_notebook)
488     data_query_notebook.add(year_tab, text="Year")
489     year_fields = ["Choice", "Year"]
490     add_query_fields(year_tab, year_fields, {"Year": validate_year},
491                     status_label, "Year")
492
493
494 def reset_database(engine):
495     Base.metadata.drop_all(engine)
496     Base.metadata.create_all(engine)
497
498 def initialize_gui():
499     window = tk.Tk()
500     window.title("University Program Evaluation System")
501
502     menubar = tk.Menu(window)
503     window.config(menu=menubar)
504
505     settings_menu = tk.Menu(menubar, tearoff=0)
506     menubar.add_cascade(label="Settings", menu=settings_menu)
507
508     settings_menu.add_command(label="Reset Database", command=reset_database)
509
510     # status Label for showing messages
511     status_label = tk.Label(window, text="", font="Helvetica 12", fg="red")
512     status_label.pack(pady=(5, 10))
513
514     main_notebook = ttk.Notebook(window)
515     main_notebook.pack(expand=True, fill="both", padx=10, pady=10)
516
517     # Setup Data Entry Tab
518     setup_data_entry_tab(main_notebook, status_label)
519
520     # Setup Data Query Tab
521     setup_data_query_tab(main_notebook, status_label)
522
523     window.mainloop()
524
525
526 def initialize_db(db):
527     global DB
528     DB = db
529
530     # Department
531     db.add_department("Cox School of Business", "BIZ")
532     db.add_department("Lyle School of Engineering", "ENG")
533
534     # Faculty
535     db.add_faculty("Vishal Ahuja", "vishal.ahuja@smu.edu", "associate", "BIZ")
536     db.add_faculty("Amy Altizer", "amy.altizer@smu.edu", "adjunct", "BIZ")
537     db.add_faculty("Thomas Barry", "thomas.barry@smu.edu", "full", "BIZ")
538     db.add_faculty("Wendy Bradley", "wendy.bradley@smu.edu", "assistant", "BIZ")
539
540     db.add_faculty("Frank Coyle", "frank.coyle@smu.edu", "associate", "ENG")
541     db.add_faculty("Qiguo Jing", "qiguo.jing@smu.edu", "adjunct", "ENG")
542     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")

```

```

611 db.assign_course_to_program(3, "BIZ2100")
612 for i in range(9, 11):
613     isSubObjective = True if i % 2 == 0 else False
614     if isSubObjective:
615         options = [j for j in range(9, i)]
616         # id, description, parent_id (optional)
617         db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
618         # course_id, objective_id, program_id
619         db.assign_objective_to_course("BIZ2100", i, 3)
620     else:
621         db.add_learning_objective(i, dummy_text[int(random() * 3)])
622 db.add_course("BIZ2200", "Intermediate Accounting",
623             "Intermediate class for all things Accounting", "BIZ")
624 db.assign_course_to_program(2, "BIZ2200")
625 for i in range(11, 13):
626     isSubObjective = True if i % 2 == 0 else False
627     if isSubObjective:
628         options = [j for j in range(11, i)]
629         # id, description, parent_id (optional)
630         db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
631         # course_id, objective_id, program_id
632         db.assign_objective_to_course("BIZ2200", i, 2)
633     else:
634         db.add_learning_objective(i, dummy_text[int(random() * 3)])
635 db.add_course("BIZ3000", "Advanced Finance",
636             "Advanced class for all things Finance", "BIZ")
637 db.assign_course_to_program(1, "BIZ3000")
638 for i in range(13, 15):
639     isSubObjective = True if i % 2 == 0 else False
640     if isSubObjective:
641         options = [j for j in range(13, i)]
642         # id, description, parent_id (optional)
643         db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
644         # course_id, objective_id, program_id
645         db.assign_objective_to_course("BIZ3000", i, 1)
646     else:
647         db.add_learning_objective(i, dummy_text[int(random() * 3)])
648 db.add_course("BIZ3100", "Advanced Marketing",
649             "Advanced class for all things Marketing", "BIZ")
650 db.assign_course_to_program(3, "BIZ3100")
651 for i in range(15, 17):
652     isSubObjective = True if i % 2 == 0 else False
653     if isSubObjective:
654         options = [j for j in range(15, i)]
655         # id, description, parent_id (optional)
656         db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
657         # course_id, objective_id, program_id
658         db.assign_objective_to_course("BIZ3100", i, 3)
659     else:
660         db.add_learning_objective(i, dummy_text[int(random() * 3)])
661 db.add_course("BIZ3200", "Advanced Accounting",
662             "Advanced class for all things Accounting", "BIZ")
663 db.assign_course_to_program(2, "BIZ3200")
664 for i in range(17, 19):
665     isSubObjective = True if i % 2 == 0 else False
666     if isSubObjective:
667         options = [j for j in range(17, i)]
668         # id, description, parent_id (optional)
669         db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
670         # course_id, objective_id, program_id
671         db.assign_objective_to_course("BIZ3200", i, 2)
672     else:
673         db.add_learning_objective(i, dummy_text[int(random() * 3)])
674
675 db.add_course("ENG1000", "Intro to Computer Science",
676             "Introduction to all things Computer Science", "ENG")
677 db.assign_course_to_program(4, "ENG1000")
678 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)]

```



```

747         # id, description, parent_id (optional)
748         db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
749         # course_id, objective_id, program_id
750         db.assign_objective_to_course("ENG2200", i, 6)
751     else:
752         db.add_learning_objective(i, dummy_text[int(random() * 3)])
753 db.add_course("ENG3000", "Advanced Computer Science",
754             "Advanced class for all things Computer Science", "ENG")
755 db.assign_course_to_program(4, "ENG3000")
756 for i in range(31, 33):
757     isSubObjective = True if i % 2 == 0 else False
758     if isSubObjective:
759         options = [j for j in range(31, i)]
760         # id, description, parent_id (optional)
761         db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
762         # course_id, objective_id, program_id
763         db.assign_objective_to_course("ENG3000", i, 4)
764     else:
765         db.add_learning_objective(i, dummy_text[int(random() * 3)])
766 db.add_course("ENG3100", "Advanced Computer Engineering",
767             "Advanced class for all things Computer Engineering", "ENG")
768 db.assign_course_to_program(5, "ENG3100")
769 for i in range(33, 35):
770     isSubObjective = True if i % 2 == 0 else False
771     if isSubObjective:
772         options = [j for j in range(33, i)]
773         # id, description, parent_id (optional)
774         db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
775         # course_id, objective_id, program_id
776         db.assign_objective_to_course("ENG3100", i, 5)
777     else:
778         db.add_learning_objective(i, dummy_text[int(random() * 3)])
779 db.add_course("ENG3200", "Advanced Creative Computing",
780             "Advanced class for all things Creative Computing", "ENG")
781 db.assign_course_to_program(6, "ENG3200")
782 for i in range(35, 37):
783     isSubObjective = True if i % 2 == 0 else False
784     if isSubObjective:
785         options = [j for j in range(35, i)]
786         # id, description, parent_id (optional)
787         db.add_learning_objective(i, dummy_text[int(random() * 3)], options[int(random() * len(options))])
788         # course_id, objective_id, program_id
789         db.assign_objective_to_course("ENG3200", i, 6)
790     else:
791         db.add_learning_objective(i, dummy_text[int(random() * 3)])
792
793 # Section
794 semester = ["Fall", "Spring", "Summer"]
795 year = [21, 22, 23, 24, 25]
796 eval = ["Exam", "Homework", "Participation"]
797 # db.add_section(200, "Fall", 23, "BIZ1000", 3, 40) # number, semester, year, course_id, instructor_id, enrollment_count
798 # db.add_section_evaluation(1, 1, "Good", 20) # section_id, objective_id, evaluation_method, students_met
799 secid = 0
800 for i in range(0, 51, 10):
801     secnum = int(random() * 10) + i
802     sem = semester[int(random() * 3)]
803     yr = year[int(random() * 5)]
804     cid = "BIZ1000"
805     iid = int(random() * 4) + 1
806     numstudents = int(random() * 50)
807     db.add_section(secnum, sem, yr, cid, iid, numstudents)
808     secid += 1
809     for j in range(1, 3):
810         ev = eval[int(random() * len(eval))]
811         numEvStudents = int(random() * numstudents)
812         db.add_section_evaluation(secid, j, ev, numEvStudents)
813 for i in range(0, 51, 10):
814     secnum=int(random() * 10) + i

```

```

815     sem = semester[int(random() * 3)]
816     yr = year[int(random() * 5)]
817     cid = "BIZ1100"
818     iid = int(random() * 4) + 1
819     numstudents = int(random() * 50)
820     db.add_section(secnum, sem, yr, cid, iid, numstudents)
821     secid += 1
822     for j in range(3, 5):
823         ev = eval[int(random() * len(eval))]
824         numEvStudents = int(random() * numstudents)
825         db.add_section_evaluation(secid, j, ev, numEvStudents)
826 for i in range(0, 51, 10):
827     secnum = int(random() * 10) + i
828     sem = semester[int(random() * 3)]
829     yr = year[int(random() * 5)]
830     cid = "BIZ1200"
831     iid = int(random() * 4) + 1
832     numstudents = int(random() * 50)
833     db.add_section(secnum, sem, yr, cid, iid, numstudents)
834     secid += 1
835     for j in range(5, 7):
836         ev = eval[int(random() * len(eval))]
837         numEvStudents = int(random() * numstudents)
838         db.add_section_evaluation(secid, j, ev, numEvStudents)
839 for i in range(0, 51, 10):
840     secnum = int(random() * 10) + i
841     sem = semester[int(random() * 3)]
842     yr = year[int(random() * 5)]
843     cid = "BIZ2000"
844     iid = int(random() * 4) + 1
845     numstudents = int(random() * 50)
846     db.add_section(secnum, sem, yr, cid, iid, numstudents)
847     secid += 1
848     for j in range(7, 9):
849         ev = eval[int(random() * len(eval))]
850         numEvStudents = int(random() * numstudents)
851         db.add_section_evaluation(secid, j, ev, numEvStudents)
852 for i in range(0, 51, 10):
853     secnum = int(random() * 10) + i
854     sem = semester[int(random() * 3)]
855     yr = year[int(random() * 5)]
856     cid = "BIZ2100"
857     iid = int(random() * 4) + 1
858     numstudents = int(random() * 50)
859     db.add_section(secnum, sem, yr, cid, iid, numstudents)
860     secid += 1
861     for j in range(9, 11):
862         ev = eval[int(random() * len(eval))]
863         numEvStudents = int(random() * numstudents)
864         db.add_section_evaluation(secid, j, ev, numEvStudents)
865 for i in range(0, 51, 10):
866     secnum = int(random() * 10) + i
867     sem = semester[int(random() * 3)]
868     yr = year[int(random() * 5)]
869     cid = "BIZ2200"
870     iid = int(random() * 4) + 1
871     numstudents = int(random() * 50)
872     db.add_section(secnum, sem, yr, cid, iid, numstudents)
873     secid += 1
874     for j in range(11, 13):
875         ev = eval[int(random() * len(eval))]
876         numEvStudents = int(random() * numstudents)
877         db.add_section_evaluation(secid, j, ev, numEvStudents)
878 for i in range(0, 51, 10):
879     secnum = int(random() * 10) + i
880     sem = semester[int(random() * 3)]
881     yr = year[int(random() * 5)]
882     cid = "BIZ3000"

```

```

883         iid = int(random() * 4) + 1
884         numstudents = int(random() * 50)
885         db.add_section(secnum, sem, yr, cid, iid, numstudents)
886         secid += 1
887         for j in range(13, 15):
888             ev = eval[int(random() * len(eval))]
889             numEvStudents = int(random() * numstudents)
890             db.add_section_evaluation(secid, j, ev, numEvStudents)
891     for i in range(0, 51, 10):
892         secnum = int(random() * 10) + i
893         sem = semester[int(random() * 3)]
894         yr = year[int(random() * 5)]
895         cid = "BIZ3100"
896         iid = int(random() * 4) + 1
897         numstudents = int(random() * 50)
898         db.add_section(secnum, sem, yr, cid, iid, numstudents)
899         secid += 1
900         for j in range(15, 17):
901             ev = eval[int(random() * len(eval))]
902             numEvStudents = int(random() * numstudents)
903             db.add_section_evaluation(secid, j, ev, numEvStudents)
904     for i in range(0, 51, 10):
905         secnum = int(random() * 10) + i
906         sem = semester[int(random() * 3)]
907         yr = year[int(random() * 5)]
908         cid = "BIZ3200"
909         iid = int(random() * 4) + 1
910         numstudents = int(random() * 50)
911         db.add_section(secnum, sem, yr, cid, iid, numstudents)
912         secid += 1
913         for j in range(17, 19):
914             ev = eval[int(random() * len(eval))]
915             numEvStudents = int(random() * numstudents)
916             db.add_section_evaluation(secid, j, ev, numEvStudents)
917     for i in range(0, 51, 10):
918         secnum = int(random() * 10) + i
919         sem = semester[int(random() * 3)]
920         yr = year[int(random() * 5)]
921         cid = "ENG1000"
922         iid = int(random() * 4) + 1
923         numstudents = int(random() * 50)
924         db.add_section(secnum, sem, yr, cid, iid, numstudents)
925         secid += 1
926         for j in range(19, 21):
927             ev = eval[int(random() * len(eval))]
928             numEvStudents = int(random() * numstudents)
929             db.add_section_evaluation(secid, j, ev, numEvStudents)
930     for i in range(0, 51, 10):
931         secnum = int(random() * 10) + i
932         sem = semester[int(random() * 3)]
933         yr = year[int(random() * 5)]
934         cid = "ENG1100"
935         iid = int(random() * 4) + 1
936         numstudents = int(random() * 50)
937         db.add_section(secnum, sem, yr, cid, iid, numstudents)
938         secid += 1
939         for j in range(21, 23):
940             ev = eval[int(random() * len(eval))]
941             numEvStudents = int(random() * numstudents)
942             db.add_section_evaluation(secid, j, ev, numEvStudents)
943     for i in range(0, 51, 10):
944         secnum = int(random() * 10) + i
945         sem = semester[int(random() * 3)]
946         yr = year[int(random() * 5)]
947         cid = "ENG1200"
948         iid = int(random() * 4) + 1
949         numstudents = int(random() * 50)
950         db.add_section(secnum, sem, yr, cid, iid, numstudents)

```

```

951         secid += 1
952     for j in range(23, 25):
953         ev = eval[int(random() * len(eval))]
954         numEvStudents = int(random() * numstudents)
955         db.add_section_evaluation(secid, j, ev, numEvStudents)
956 for i in range(0, 51, 10):
957     secnum = int(random() * 10) + i
958     sem = semester[int(random() * 3)]
959     yr = year[int(random() * 5)]
960     cid = "ENG2000"
961     iid = int(random() * 4) + 1
962     numstudents = int(random() * 50)
963     db.add_section(secnum, sem, yr, cid, iid, numstudents)
964     secid += 1
965     for j in range(25, 27):
966         ev = eval[int(random() * len(eval))]
967         numEvStudents = int(random() * numstudents)
968         db.add_section_evaluation(secid, j, ev, numEvStudents)
969 for i in range(0, 51, 10):
970     secnum = int(random() * 10) + i
971     sem = semester[int(random() * 3)]
972     yr = year[int(random() * 5)]
973     cid = "ENG2100"
974     iid = int(random() * 4) + 1
975     numstudents = int(random() * 50)
976     db.add_section(secnum, sem, yr, cid, iid, numstudents)
977     secid += 1
978     for j in range(27, 29):
979         ev = eval[int(random() * len(eval))]
980         numEvStudents = int(random() * numstudents)
981         db.add_section_evaluation(secid, j, ev, numEvStudents)
982 for i in range(0, 51, 10):
983     secnum = int(random() * 10) + i
984     sem = semester[int(random() * 3)]
985     yr = year[int(random() * 5)]
986     cid = "ENG2200"
987     iid = int(random() * 4) + 1
988     numstudents = int(random() * 50)
989     db.add_section(secnum, sem, yr, cid, iid, numstudents)
990     secid += 1
991     for j in range(29, 31):
992         ev = eval[int(random() * len(eval))]
993         numEvStudents = int(random() * numstudents)
994         db.add_section_evaluation(secid, j, ev, numEvStudents)
995 for i in range(0, 51, 10):
996     secnum = int(random() * 10) + i
997     sem = semester[int(random() * 3)]
998     yr = year[int(random() * 5)]
999     cid = "ENG3000"
1000     iid = int(random() * 4) + 1
1001     numstudents = int(random() * 50)
1002     db.add_section(secnum, sem, yr, cid, iid, numstudents)
1003     secid += 1
1004     for j in range(31, 33):
1005         ev = eval[int(random() * len(eval))]
1006         numEvStudents = int(random() * numstudents)
1007         db.add_section_evaluation(secid, j, ev, numEvStudents)
1008 for i in range(0, 51, 10):
1009     secnum = int(random() * 10) + i
1010     sem = semester[int(random() * 3)]
1011     yr = year[int(random() * 5)]
1012     cid = "ENG3100"
1013     iid = int(random() * 4) + 1
1014     numstudents = int(random() * 50)
1015     db.add_section(secnum, sem, yr, cid, iid, numstudents)
1016     secid += 1
1017     for j in range(33, 35):
1018         ev = eval[int(random() * len(eval))]

```

```
1019         numEvStudents = int(random() * numstudents)
1020         db.add_section_evaluation(secid, j, ev, numEvStudents)
1021     for i in range(0, 51, 10):
1022         secnum = int(random() * 10) + i
1023         sem = semester[int(random() * 3)]
1024         yr = year[int(random() * 5)]
1025         cid = "ENG3200"
1026         iid = int(random() * 4) + 1
1027         numstudents = int(random() * 50)
1028         db.add_section(secnum, sem, yr, cid, iid, numstudents)
1029         secid += 1
1030     for j in range(35, 37):
1031         ev = eval[int(random() * len(eval))]
1032         numEvStudents = int(random() * numstudents)
1033         db.add_section_evaluation(secid, j, ev, numEvStudents)
1034
1035     print('db initialized successfully')
```

## db.py

```

1  from sqlalchemy import (
2      Column,
3      Integer,
4      String,
5      ForeignKey,
6      Text,
7      create_engine,
8      text,
9  )
10 from sqlalchemy.ext.declarative import declarative_base
11 from sqlalchemy.orm import relationship, sessionmaker
12
13 Base = declarative_base()
14
15
16 class Department(Base):
17     __tablename__ = "departments"
18     id = Column(Integer, primary_key=True)
19     name = Column(String(255), unique=True)
20     code = Column(String(4), unique=True)
21     faculty = relationship("Faculty", backref="department")
22     programs = relationship("Program", backref="department")
23
24
25 class Faculty(Base):
26     __tablename__ = "faculty"
27     id = Column(Integer, primary_key=True)
28     name = Column(String(255))
29     email = Column(String(255), unique=True)
30     rank = Column(String(50))
31     department_code = Column(String(4), ForeignKey("departments.code"))
32
33
34 class Program(Base):
35     __tablename__ = "programs"
36     id = Column(Integer, primary_key=True)
37     name = Column(String(255), unique=True)
38     department_code = Column(String(4), ForeignKey("departments.code"))
39     in_charge_id = Column(Integer, ForeignKey("faculty.id"))
40     courses = relationship("Course", secondary="program_courses")
41
42
43 class Course(Base):
44     __tablename__ = "courses"
45     id = Column(String(10), primary_key=True)
46     title = Column(String(255))
47     description = Column(Text)
48     department_code = Column(String(4), ForeignKey("departments.code"))
49     sections = relationship("Section", backref="course")
50
51
52 class Section(Base):
53     __tablename__ = "sections"
54     id = Column(Integer, primary_key=True)
55     number = Column(Integer)
56     semester = Column(String(20))
57     year = Column(Integer)
58     course_id = Column(String(10), ForeignKey("courses.id"))
59     instructor_id = Column(Integer, ForeignKey("faculty.id"))
60     enrollment_count = Column(Integer)
61
62
63 class LearningObjective(Base):
64     __tablename__ = "learning_objectives"
65     id = Column(String(50), primary_key=True)
66     description = Column(Text)

```

```

67     parent_id = Column(String(50), ForeignKey("learning_objectives.id"))
68     sub_objectives = relationship("LearningObjective")
69
70
71     class ProgramCourses(Base):
72         __tablename__ = "program_courses"
73         program_id = Column(Integer, ForeignKey("programs.id"), primary_key=True)
74         course_id = Column(String(10), ForeignKey("courses.id"), primary_key=True)
75
76
77     class CourseObjectives(Base):
78         __tablename__ = "course_objectives"
79         course_id = Column(String(10), ForeignKey("courses.id"), primary_key=True)
80         objective_id = Column(
81             String(50), ForeignKey("learning_objectives.id"), primary_key=True
82         )
83         program_id = Column(Integer, ForeignKey("programs.id"), primary_key=True)
84
85
86     class SectionEvaluations(Base):
87         __tablename__ = "section_evaluations"
88         section_id = Column(Integer, ForeignKey("sections.id"), primary_key=True)
89         objective_id = Column(String(50), ForeignKey("learning_objectives.id"), primary_key=True)
90         evaluation_method = Column(String(255), primary_key=True)
91         students_met = Column(Integer)
92
93
94     class SessionManager:
95         def __init__(self, database_uri):
96             self.engine = create_engine(database_uri)
97             Session = sessionmaker(bind=self.engine)
98             self.session = Session()
99
100         def add(self, entry):
101             try:
102                 self.session.add(entry)
103                 self.session.commit()
104             except:
105                 self.session.rollback()
106                 raise
107
108         def add_department(self, name, code):
109             new_department = Department(name=name, code=code)
110             self.add(new_department)
111
112         def add_faculty(self, name, email, rank, department_code):
113             new_faculty = Faculty(
114                 name=name, email=email, rank=rank, department_code=department_code
115             )
116             self.add(new_faculty)
117
118         def add_program(self, name, department_code, in_charge_id):
119             new_program = Program(
120                 name=name, department_code=department_code, in_charge_id=in_charge_id
121             )
122             self.add(new_program)
123
124         def add_course(self, id, title, description, department_code):
125             new_course = Course(
126                 id=id, title=title, description=description, department_code=department_code
127             )
128             self.add(new_course)
129
130         def add_section(
131             self, number, semester, year, course_id, instructor_id, enrollment_count
132         ):
133             new_section = Section(
134                 number=number,

```

```

135         semester=semester,
136         year=year,
137         course_id=course_id,
138         instructor_id=instructor_id,
139         enrollment_count=enrollment_count,
140     )
141     self.add(new_section)
142
143     def add_learning_objective(self, id, description, parent_id=None):
144         new_objective = LearningObjective(
145             id=id, description=description, parent_id=parent_id
146         )
147         self.add(new_objective)
148
149     def assign_course_to_program(self, program_id, course_id):
150         new_assignment = ProgramCourses(program_id=program_id, course_id=course_id)
151         self.add(new_assignment)
152
153     def assign_objective_to_course(self, course_id, objective_id, program_id):
154         new_assignment = CourseObjectives(
155             course_id=course_id, objective_id=objective_id, program_id=program_id
156         )
157         self.add(new_assignment)
158
159     def add_section_evaluation(
160         self, section_id, objective_id, evaluation_method, students_met
161     ):
162         new_evaluation = SectionEvaluations(
163             section_id=section_id,
164             objective_id=objective_id,
165             evaluation_method=evaluation_method,
166             students_met=students_met,
167         )
168         self.add(new_evaluation)
169
170     def query(self, query):
171         return [[attr for attr in row] for row in self.session.execute(text(query))]
172
173     # List all of its programs
174     def get_department_programs_by_name(self, department_name):
175         return self.query(
176             f"SELECT p.name "
177             f"FROM departments AS d "
178             f"JOIN programs AS p "
179             f"ON d.code = p.department_code "
180             f"WHERE d.name= '{department_name}'"
181         )
182
183     # List all of its faculty (including what program each faculty is in charge of, if there is one)
184     def get_department_faculty_by_name(self, department_name):
185         return self.query(
186             f"SELECT f.name, p.name "
187             f"FROM departments AS d "
188             f"JOIN faculty AS f "
189             f"ON d.code = f.department_code "
190             f"LEFT JOIN programs AS p "
191             f"ON f.id = p.in_charge_id "
192             f"WHERE d.name = '{department_name}'"
193         )
194
195     # List all the courses, together with the objectives/sub-objectives association with year
196
197     def get_program_courses_by_name(self, program_name):
198         return self.query(
199             f"SELECT c.title, lo.description "
200             f"FROM courses AS c "
201             f"JOIN program_courses AS pc "
202             f"ON c.id = pc.course_id "

```



```

203         f"JOIN programs AS p "
204         f"ON p.id = pc.program_id "
205         f"LEFT JOIN course_objectives AS co "
206         f"ON co.course_id = c.id "
207         f"LEFT JOIN learning_objectives AS lo "
208         f"ON co.objective_id = lo.id "
209         f"WHERE p.name = '{program_name}'"
210     )
211
212     # List all of the objectives
213     def get_program_objectives_by_name(self, program_name):
214         return self.query(
215             f"SELECT DISTINCT lo.description "
216             f"FROM courses AS c "
217             f"JOIN program_courses AS pc "
218             f"ON c.id = pc.course_id "
219             f"JOIN programs AS p "
220             f"ON p.id = pc.program_id "
221             f"JOIN course_objectives AS co "
222             f"ON co.course_id = c.id "
223             f"JOIN learning_objectives AS lo "
224             f"ON co.objective_id = lo.id "
225             f"WHERE p.name = '{program_name}'"
226         )
227
228     # List all of the evaluation results for each objective/sub-objective (If data for some sections has not been entered, indicate with null)
229     def get_results_by_semester(self, semester, program_name):
230         semester = semester.split(" ")
231         return self.query(
232             f"SELECT c.title, s.number, se.evaluation_method, se.students_met "
233             f"FROM courses AS c "
234             f"JOIN sections AS s "
235             f"ON c.id = s.course_id "
236             f"JOIN program_courses AS pc "
237             f"ON c.id = pc.course_id "
238             f"JOIN programs AS p "
239             f"ON p.id = pc.program_id "
240             f"LEFT OUTER JOIN section_evaluations AS se "
241             f"ON s.id = se.section_id "
242             f"WHERE s.semester = '{semester[0]}' " # semester[0] will contain "Fall" "Spring" etc
243             f"AND s.year = {semester[1]} " # semester[1] will contain year (i.e. 23, 24, etc.)
244             f"AND p.name = '{program_name}'"
245         )
246
247     # List all of the evaluation results for each objective/sub-objective
248     # Show course/section involved in evaluation, list result for each course/section, and aggregate the result to show the number of students met
249     def get_results_by_year(self, year):
250         year = year.split("-")
251         return self.query(
252             f"SELECT lo.description, c.title, s.number, se.evaluation_method, se.students_met, ROUND(se.students_met * 100.0/s.evaluation_method, 2) AS pct_met "
253             f"FROM courses AS c "
254             f"JOIN sections AS s "
255             f"ON c.id = s.course_id "
256             f"JOIN course_objectives AS co "
257             f"ON c.id = co.course_id "
258             f"JOIN learning_objectives AS lo "
259             f"ON co.objective_id = lo.id "
260             f"JOIN section_evaluations AS se "
261             f"ON s.id = se.section_id AND lo.id = se.objective_id "
262             f"WHERE (s.year = {year[0]} "
263             f"AND s.semester = 'Summer') "
264             f"OR (s.year = {year[0]} "
265             f"AND s.semester = 'Fall') "
266             f"OR (s.year = {year[1]} "
267             f"AND s.semester = 'Spring') "
268             f"GROUP BY lo.id, c.id, s.id, se.evaluation_method "
269             f"ORDER BY c.id, s.id"
270         )

```