A logo of a university of windsor

Description automatically generated with low confidence

|  |
| --- |
| **Course Code | Course Name** |
| COMP8347-2-R-2023F | Internet Appl/ Distributed Sys |

|  |
| --- |
| **Document Type** |
| Lab 4 |

|  |
| --- |
| **Professor** |
| Dr. Saja Al Mamoori |

|  |
| --- |
| Graduate Assistant |
| Ms. Amangel Bhullar |

|  |
| --- |
| **Team - Members                               Student ID** |
| Manjinder Singh                                 110097177 |
| Harbhajan Singh 110100089 |
| Karan Vishavjit 110099867 |
| Khyati Shah 110090234 |

***NOTES :*** For simplicity questions and content from lab manual 4 are mentioned in a box.

|  |
| --- |
| **Part 3: Querying the database.** |

|  |
| --- |
| 1. Open **Python Console.** In **Python console** import Django then models from *models.py.*  import django  from myappF23.models import Student, Course, Instructor, Category |

A screen shot of a computer

Description automatically generated

**Screenshot 1:** Snippet of solution 1

|  |
| --- |
| 1. Write queries to obtain the following information. Verify if your query generates the correct answer using the data entered in question 2.5. Can you use annotate for any of these queries? explain to the grader. |

The **annotate** method in Django's ORM is used to perform aggregations on your data, like counting, summing, or averaging related objects. None of the given queries require aggregations. All queries listed below are simple retrieval operations without a need for aggregation. Thus, in this specific case, there's no need to use **annotate**.

|  |
| --- |
| **2.1** Write a separate query to list all the students, instructors, courses, and categories in the db. |

print("\nAnswer 2.1 ---")

students = Student.objects.all()

print("1. Enlisting student names:- ")

for student in students:

print("-->",student.first\_name, student.last\_name)

instructors = Instructor.objects.all()

print("\n2. Enlisting instructor names:- ")

for instructor in instructors:

print("-->",instructor.first\_name, instructor.last\_name)

courses = Course.objects.all()

print("\n3. Enlisting course names:-")

for course in courses:

print("-->",course.title)

categories = Category.objects.all()

print("\n4. Enlisting course categories:-")

for category in categories:

print("-->",category.name)

A computer screen shot of a program

Description automatically generated

**Screenshot 2:** Snippet of code solution of question 2.1

A screenshot of a computer program

Description automatically generated

**Screenshot 3:** Snippet of code output of question 2.1

|  |
| --- |
| * 1. List all courses in a specific category (e.g., 'Programming') |

filter\_by\_category = Course.objects.filter(categories\_\_name='Programming')

print("\nAnswer 2.2 - Enlisting all courses of Programming category:-")

for index, course in enumerate(filter\_by\_category, start=1):

print(f"{index}. {course.title}")

A computer screen shot of a program

Description automatically generated

**Screenshot 4:** Snippet of code solution and output of question 2.2

|  |
| --- |
| * 1. List all courses taught by a specific instructor (e.g., 'John Smith) |

filter\_by\_instructor = Course.objects.filter(instructor\_\_first\_name='John', instructor\_\_last\_name='Smith')

print("\nAnswer 2.3 - Courses taught by John Smith instructor:-")

for index, course in enumerate(filter\_by\_instructor, start=1):

print(f"{index}. {course.title}")

A screen shot of a computer program

Description automatically generated

**Screenshot 5:** Snippet of code solution and output of question 2.3

|  |
| --- |
| * 1. List all courses with a price greater than $100. |

filter\_by\_price = Course.objects.filter(price\_\_gt=100)

print("\nAnswer 2.4 - Enlisting courses that are priced above $100:")

for index, course in enumerate(filter\_by\_price, start=1):

print(f"{index}. {course.title}")

A screen shot of a computer program

Description automatically generated

**Screenshot 6:** Snippet of code solution and output of question 2.4

|  |
| --- |
| 2.5 List all students enrolled in a specific course (e.g., 'Web Dev Bootcamp'). |

filter\_by\_course\_name = Student.objects.filter(course\_\_title='Web Dev Bootcamp')

print("\nAnswer 2.5 - Students enrolled in Web Dev Bootcamp course:")

for index, student in enumerate(filter\_by\_course\_name, start=1):

print(f"{index}. {student.first\_name} {student.last\_name}")

A computer screen with text on it

Description automatically generated

**Screenshot 7:** Snippet of code solution and output of question 2.5

|  |
| --- |
| 2.6 List all courses with start dates in the future. |

from datetime import datetime

future\_courses = Course.objects.filter(start\_date\_\_gt=datetime.now().date())

print("\nAnswer 2.6 - Enlisting all the courses that are scheduled in the future:-")

for index, course in enumerate(future\_courses, start=1):

print(f"{index}. {course.title}")

A computer screen shot of a program

Description automatically generated

**Screenshot 8:** Snippet of code solution and output of question 2.6

|  |
| --- |
| 2.7 **Retrieve a specific student by email (e.g., 'john@gmail.com'):** |

get\_name\_from\_email = Student.objects.get(email=’lnu8@uwindsor.ca')

print(f"\nAnswer 2.7 - Student name with email id - { get\_name\_from\_email.email} is having name { get\_name\_from\_email.first\_name} { get\_name\_from\_email.last\_name}.")

A screen shot of a computer code

Description automatically generated

**Screenshot 9:** Snippet of code solution and output of question 2.7

|  |
| --- |
| 2.8 **Retrieve a specific instructor by name (e.g., 'Saja Al-Mamoori'):** |

Inst\_Name = Instructor.objects.get(first\_name='Saja', last\_name='Al-Mamoori')

print(f"\nAnswer 2.8 – Fetched instructor name:- {Inst\_Name.first\_name} {Inst\_Name.last\_name}")

A black screen with colorful text

Description automatically generated

**Screenshot 10:** Snippet of code solution and output of question 2.8

|  |
| --- |
| 2.9 **List all courses with a specific difficulty level (e.g., 'Intermediate').** |

courses\_of\_intermediate\_level = Course.objects.filter(level='IN')

print("\nAnswer 2.9 – Enlisting courses of intermediate level:-")

for index, course in enumerate(courses\_of\_intermediate\_level, start=1):

print(f"{index}. {course.title}")

A screen shot of a computer code

Description automatically generated

**Screenshot 11:** Snippet of code solution and output of question 2.9

|  |
| --- |
| 2.10 **List all courses that have the word 'Python' in their title.** |

courses\_with\_word\_python = Course.objects.filter(title\_\_icontains='Python')

print("\nAnswer 2.10 - Enlisting courses containing 'Python' in title:-")

for index, course in enumerate(courses\_with\_word\_python, start=1):

print(f"{index}. {course.title}")

A screen shot of a computer program

Description automatically generated

**Screenshot 12:** Snippet of code solution and output of question 2.10

|  |
| --- |
| 2.11 **List all students who were born after a certain date (e.g., after January 1, 2000).** |

students\_year2000 = Student.objects.filter(date\_of\_birth\_\_gt="2000-01-01")

print("\nAnswer 2.11 – Enlisting students born after January 1, 2000:- ")

for index, student in enumerate(students\_year2000, start=1):

print(f"{index}. {student.first\_name} {student.last\_name}")

A screen shot of a computer code

Description automatically generated

**Screenshot 13:** Snippet of code solution and output of question 2.11

|  |
| --- |
| 2.12 **List all courses that started after a specific date (e.g., after January 1, 2023).** |

courses2023 = Course.objects.filter(start\_date\_\_gt="2023-01-01")

print("\nAnswer 2.12 – Enlisting courses that started after January 1, 2023:-")

for index, course in enumerate(courses2023, start=1):

print(f"{index}. {course.title}")

A computer screen with white and green text

Description automatically generated

**Screenshot 14:** Snippet of code solution and output of question 2.12