University of Hertfordshire

Data Mining and Discovery

SQL Assignment

Cristina Baron Suarez cb24ack@herts.ac.uk

March 2, 2025

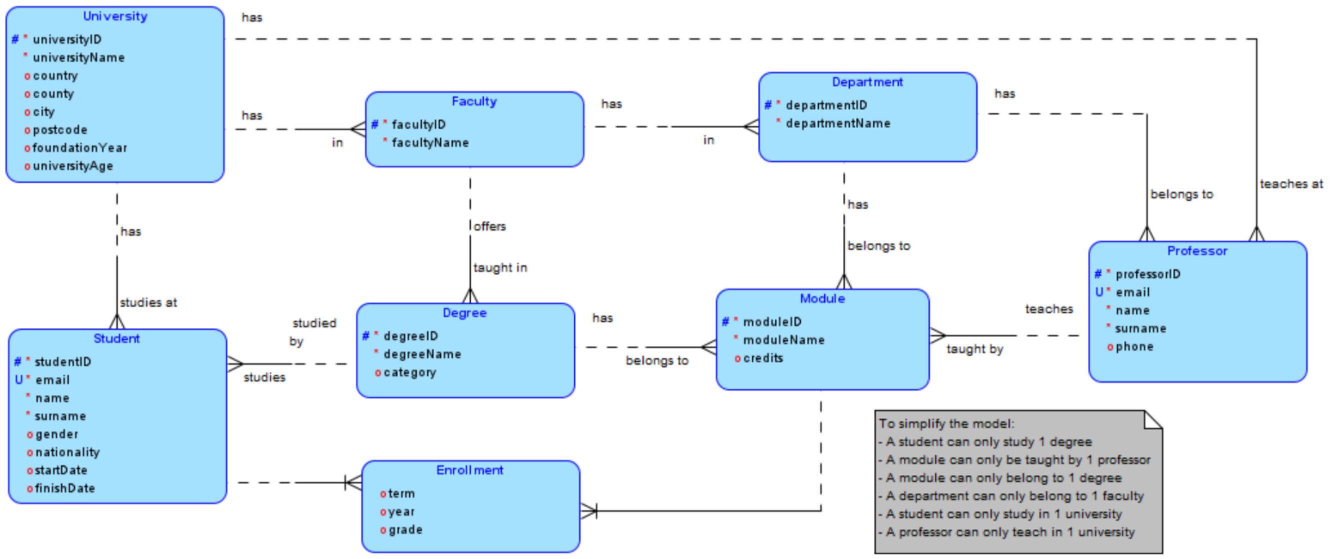
Contents

1 Introduction

The purpose of this assignment is to generate a database using SQLite software. For that, I chose to recreate the database of a university. I am specifically creating realistic data for the University of Hertfordshire. This means that I have taken into account the creation of faculties, departments, modules, degrees, students, professors, and, obviously, the university.

2 Database Schema

SQL Developer Data Modeler has been used in order to generate the database schema shown below.



To simplify the model:
- A student can only study 1 degree
- A module can only be taught by 1 professor
- A module can only belong to 1 degree
- A department can only belong to 1 faculty
- A student can only study in 1 university
- A professor can only teach in 1 university

We can appreciate the following relationships:

- A faculty must belong to a university, and a university can have several faculties.
- A department must belong to a faculty, and a faculty can have several departments.
- A professor must belong to a department, and a department can have several professors.
- A module must be taught by one professor, which will belong to the same department as the module. Furthermore, a professor can teach several modules, as long as they belong to their department.
- A module must also belong to a degree, and this degree can have several modules. This degree must belong to the same faculty to which the module belongs to (related through the department).
- The student is the one who studies the degree, and they can have several classmates. On the contrary, students belong to a university.
- As a student is going to have several modules and a module is going to be studied by several students, I have decided to build a weak entity called "Enrollment". This table will share the primary key of the students and the modules so that we can identify unique instances of a student's enrollment in a particular module.

2.1 Attribute Constraints

Each table has a column that identifies them, called as *table_name*ID, which is the primary key of the entity. On the other hand, in the 'Student' and 'Professor' tables we can find the unique constraint 'email', since an email address can only belong to one person. Furthermore, I have set non-null constraints on fields that I consider esential.

3 Ethical and Data Privacy Considerations

To ensure ethical and data privacy, the university postcode will only show the first four digits, and the professor's phone number will only be partially visible if a record exists.

```
postcode = ['AL10']
professor_phone.append('+44 7********' + str(random.randint(00, 99)))
```

4 Data Types

We can find several columns in the database that can be assigned as any of the NOIR attributes.

4.1 Nominal Data

The columns with nominal data in them are the following:

- **Gender**, in the student table ('Male' or 'Female').
- **Nationality**, in the student table.
- **Faculty Name**, in the faculty table.
- **Module Name**, in the module table.
- **Degree Name**, in the degree table.

4.2 Ordinal Data

The column with ordinal data in it is the following one:

- **Category**, in the Degree's table ('Undergraduate' or 'Postgraduate').

4.3 Interval Data

The columns with interval data in them are the following:

- **University Age**, in the University table. This attribute represents the actual year minus the foundation year. That is, the number zero would not mean 'no age available'.
- **Credits**, in the Module's table, because a module can have 0 credits.

4.4 Ratio Data

The column with the ratio data in it is the following one:

- **Grade**, in the Enrollment table, because no student can score a zero in a subject.

5 The logic behind the dataframes

In this section we will explain the logic behind the creation of the data. See appendix A for more details.

5.1 University table

In the university table there is only one row of information - the University of Hertfordshire. In this model, we consider that we do not know any information about the rest of existing universities.

5.2 Student table

We generate 1000 students, assigning them an ID that ranges from 1 to 1000, and their gender, either 'male' or 'female'. Once we know this, we assign them a name from a series of names taking into account their gender. Then, we assign them a surname and a nationality - also from a battery of possibilities. In order to generate null data, we create a mask for the gender and the nationality. 20% of the students will not have this information available. Furthermore, the student's email will be their student ID appened to '@gmail.com'.

On the contrary, one of the 'mistakes' that I have decided to include in the database, in order to make the data more realistic, is the end date for the student's studies. As we will see in the enrollment table, a student can either study an undergraduate, which is two years long, or a postgraduate, which is one year long. However, when we generate the start date, the end date is calculated by adding 3 years to it since it is the average amount of time to finish an undergraduate degree. Only an end date that is not in the future will be saved.

In addition, here we also generate a battery of degree names, and then create a dictionary that will assign an ID to each degree name. Then, we assign each student to a random degree. For that, we create another dictionary that will have as keys the students' ID and as value the degrees' IDs.

5.3 Faculty table

Based on the created degrees, we have decided that it is best to generate seven faculties. Then, we create a dictionary that assigns each faculty (keys) to a list of degress (values). We also established that these faculties belong to the University of Hertfordshire.

5.4 Degree table

As mentioned above, in this step we decide whether a student is studying an undergraduate or a post-graduate degree. In addition, we obtain the faculty in which each degree is studied, and we also decide whether a certain degree belongs to an undergraduate or postgraduate program.

5.5 Module table

A list of template string for module names is created, and then we iterate through each degree and the list of module templates, replacing '{degree}' for the actual degree name. Finally, we assign an ID to the module name in order to create a dictionary that would relate the ID to its name.

```
module_templates = ["Introduction to {degree}",
                    "{degree} Fundamentals",
                    "Advanced Topics in {degree}",
                    "{degree} Project",
                    "Research Methods in {degree}",
                    "Ethics and Professional Practice in {degree}"]
```

```
module_dict = {}
module_id = 1
for degree in degrees:
    for template in module_templates:
        module_name = template.format(degree = degree)
        module_dict[module_id] = module_name
        module_id += 1
```

To properly structure the assignment of modules to each degree, we start by defining a dictionary that will relate degree to the modules assigned to it. We initiate a counter for the module IDs. Then, for each degree, we determine how many modules it requires and assign a unique ID to each one, storing these IDs in a list associated with the degree in the dictionary. This process ensures that each degree is correctly linked to its relative modules.

```
degree_module = {}
module_id = 1
for deg_id in range(1, len(degrees) + 1):
    modules_for_degree = []
    for _ in range(len(module_templates)):
        modules_for_degree.append(module_id)
        module_id += 1
    degree_module[deg_id] = modules_for_degree
```

To be able to continue the creation of this dataframe, we first need to create 'Professor' and 'Department'.

5.5.1 Professor table

As we did in the student's table, we generate a couple of combinations of names and surnames, and we create as many ordered IDs as generated names. Simultaniously, the email will be the whole name of the professor with '@gmail.com' appended.

As mentioned in the 'Ethical and Data Privacy Considerations' section, in order to generate the professor's phone number, we only let the first and last two digits of the phone number be visible. These digits are generated randomly. In addition, to make the data look more realistic, 30% of the phone numbers will be set to null.

5.5.2 Department table

Based on the modules, degrees and faculties created, I have generated several departments. The names are included as values in a dictionary in which they are assigned an ID as their key. Then, we create another dictionary that will assign these department IDs (key) to their corresponding list of module IDs (values).

Secondly, we also randomly assign each professor to a unique department.

```
dept_ids = list(range(1, len(departments) + 1))
prof_ids = list(range(1, len(professor_id) + 1))
random.shuffle(prof_ids)

dic_professor_department = {}
for i in range(len(dept_ids)):
    dic_professor_department[prof_ids[i]] = dept_ids[i]
for i in range(len(dept_ids), len(prof_ids)):
    dic_professor_department[prof_ids[i]] = random.choice(dept_ids)
```

Followed by, we randomly assign modules to the professors. Both must belong to the same department.

```
dic_module_professor = {}
for mod_id in range(1, 133):
    department_id = next(dept for dept, mods in dict_department_modules.items()
     if mod_id in mods)
    eligible_professors = [p_id for p_id, dept_id in dic_professor_department.
   items() if dept_id == department_id]
    dic_module_professor[mod_id] = random.choice(eligible_professors)
```

We also create a dictionary that will assign a list of department IDs (values) to each faculty ID (keys).

5.5.3 Module table continuation

Once we have the foreign keys 'professorID' and 'departmentID', we can assign modules which will be taught by teachers, considering that both must belong to the same department.

```
degree_ids = []
department_ids = []
for mod_id in module_dict.keys():
    deg_id = next((d_id for d_id, mods in degree_module.items() if mod_id in
   mods), None)
    dep_id = next((dpt_id for dpt_id, mods in dict_department_modules.items()
   if mod_id in mods), None)
    degree_ids.append(deg_id)
    department_ids.append(dep_id)

professor_ids = [dic_module_professor.get(m_id, None) for m_id in module_dict]
```

5.6 Enrollment table

To create the enrollment dataframe, we need to obtain information from the students and the dataframe of the modules. From the students, we need to get their degree category (undergraduate or postgraduate) and their finish date, and from the module's dataframe we need the module IDs.

Another one of the 'mistakes' we have included in the database so that the data seem more real is that, for those students who do not have a finish date of studies, they are not going to be awarded grades, which is why we have decided not to include them in the dataframe. However, for those students who have completed their studies, we assign the grades as follows:

- Undergraduate: Randomly assign a grade between 50 and 100 for each module, considering that it took them two years to finish (12 modules, 6 per year, 3 per term).
- Postgraduate: Randomly assign a grade between 50 and 100 for each module, considering that it took them one year to finish (12 modules, 6 per term).

After this, for each student, we would have assigned each of their modules a year, a term, and a grade.

```python
for _, row in student.iterrows():
    st_id = row['studentID']
    deg_cat = dic_degree_category.get(st_id, 'Undergraduate').lower()
    fdate = row['finishDate']
    module_ids = module['moduleID'].tolist()

    if pd.notnull(fdate):
        if deg_cat == 'undergraduate':
            for i, m_id in enumerate(module_ids[:12]):
                year = 1 if i < 6 else 2   # Ordinal Data
                term = 1 if i % 6 < 3 else 2
                grade = np.random.uniform(50, 100)
                enrollment_rows.append({
                    'term': term, 'year': year, 'grade': round(grade, 2),
                    'studentID': st_id, 'moduleID': m_id
                })
        elif deg_cat == 'postgraduate':
            for i, m_id in enumerate(module_ids[:12]):
                year = 1
                term = 1 if i < 6 else 2
                grade = np.random.uniform(50, 100)
                enrollment_rows.append({
                    'term': term, 'year': year, 'grade': round(grade, 2),
                    'studentID': st_id, 'moduleID': m_id
                })
```

A Python code to create the data of the database

```
import numpy as np
import pandas as pd
import random

# Number of samples
n = 1000

## 'University' table
university_id = [1]
university_name = ['University of Hertfordshire']
country = ['England']
county = ['Hertfordshire']
city = ['Hatfield']
postcode = ['AL10'] # less specific for data privacy reasons
foundation_year = [1952]
universityAge = pd.Timestamp.now().year - 1952  # Interval Data

# Create the university dataframe
university = pd.DataFrame({
    'universityID': university_id,
    'universityName': university_name,
    'country': country,
    'county': county,
    'city': city,
    'postcode': postcode,
    'foundationYear': foundation_year,
    'universityAge': universityAge
})

## 'Student' table
# Generate a list of numbers from 1 to n in order
student_id = np.arange(1, n + 1)

# Randomly generate gender 'Male' or 'Female'
gender = np.random.choice(['Male', 'Female'], n)  # Nominal data

# Depending on the gender, the student will have either a female or a male name
first_name = []
for i in range(n):
    if gender[i] == 'Male':
        first_name.append(np.random.choice([
            'Carlos', 'Pedro', 'Alvaro', 'Jose', 'Jairo', 'Mateo', 'Antonio', '
   Ivan', 'Oscar', 'Ruben', 'Elmar', 'David', 'Rodrigo', 'Jose', 'Miguel', '
   Ignacio', 'Danila', 'Luca', 'Helmut', 'Guillermo', 'William', 'Elio', 'Luis'
   , 'Rafael', 'Javier', 'Eduardo', 'Juan', 'James', 'Hugo', 'Charles', 'Peter'
   , 'Hariz', 'Pir', 'Nouman', 'Ghamees', 'Michael', 'Nick', 'Louis', 'Raphael'
   , 'Joan', 'Eric', 'Adrian', 'Sergio', 'Thomas'
        ], 1)[0])
    else:
        first_name.append(np.random.choice([
            'Pia', 'Marta', 'Marina', 'Cristina', 'Ana', 'Maria', 'Lucia', '
   Paula', 'Esther', 'Luisa', 'Elena', 'Julia', 'Martha', 'Taylor', 'Christine'
   , 'Lucy', 'Paola', 'Khadija', 'Sana', 'Kinza', 'Rena', 'Carmen', 'Mar', '
   Samantha', 'Selena', 'Sabrina', 'Tate', 'Madison', 'Marie', 'Ariana', 'Ava',
    'Bebe', 'Camila', 'Hailee', 'Kate', 'Olivia'
        ], 1)[0])

# Randomly generate last name
surname = np.random.choice([
    'Garcia', 'Rodriguez', 'Martinez', 'Hernandez', 'Lopez', 'Gonzalez', 'Perez
    ', 'Sanchez', 'Bonvie', 'Benito', 'Rojano', 'Suarez', 'Baron', 'Prieto', '
    Peralta', 'Escribano', 'Campos', 'Carrobles', 'Gomez', 'Castro', 'Ruiz', '
    Cordon', 'Bautista', 'Ortiz', 'Arco', 'Montoro', 'Ramirez', 'Salas', '
    Paakkinen', 'Hutter', 'Costa', 'Rubiales', 'Delgado', 'Zaragoza', 'Correal',
     'Caro', 'Roy'
], n)

# Randomly generate nationality
nationality = np.random.choice([
    'USA', 'China', 'India', 'Pakistan', 'Spain', 'Portugal', 'Germany', '
    England', 'Nigeria', 'Ireland', 'Wales', 'Scotland', 'Norway', 'Finland', '
    Russia', 'Mexico', 'Peru', 'France', 'Argentina', 'Australia'
], n)

# 20% of the students will have NULL in the attibute 'gender' and 'nationality'
maskGender = np.random.rand(n) < 0.2
maskNationality = np.random.rand(n) < 0.2
gender[maskGender] = ''
nationality[maskNationality] = ''

# Generate student email based on the studentID
email = []
for i in range(n):
    email.append(str(student_id[i]) + '@gmail.com')

# Generate start and finish date of the student's degree
start_date = []
finish_date = []
for i in range(n):
    # Randomly generate start date (YYYY-MM-DD)
    year = str(np.random.randint(1960, 2024))
    month = str(np.random.randint(1, 13)).zfill(2)
    day = str(np.random.randint(1, 29)).zfill(2)
    start_date.append(pd.to_datetime(f"{year}-{month}-{day}"))
    # if start_date was more than 4 years ago, add 3 years to the start date
    if start_date[i] < pd.to_datetime('2020-09-01'):
        finish_date.append(start_date[i] + pd.DateOffset(years = 3))
    else:
        finish_date.append(None)

# Randomly generate degree names
degrees = ['Computer Science', 'Data Science', 'Artificial Intelligence', '
    Engineering', 'Mathematics', 'Physics', 'Chemistry', 'Biology', 'Medicine',
    'Economics', 'Business', 'Law', 'Psychology', 'Accounting and Finance', '
    Marketing', 'History', 'Geography', 'Philosophy', 'Sociology', 'Criminology'
    , 'Education', 'Politics']

# Assigns an ID to each degree
degree_dict = {degree: i + 1 for i, degree in enumerate(degrees)}
```

```python
# Assign each student to a random degree
dic_student_degree = {student: np.random.choice(list(degree_dict.keys())) for
   student in student_id}

# Create the student dataframe
student = pd.DataFrame({
    'studentID': student_id,
    'email': email,
    'name': first_name,
    'surname': surname,
    'gender': gender,
    'nationality': nationality,
    'startDate': start_date,
    'finishDate': finish_date,
    'degreeID': [degree_dict[degree] for degree in dic_student_degree.values()
   ],
    'universityID': university_id * n
})

## 'Faculty' table
faculties = {
    1: 'Faculty of Computer Science and Engineering',
    2: 'Faculty of Natural Sciences',
    3: 'Faculty of Medicine and Health Sciences',
    4: 'Faculty of Economics and Business',
    5: 'Faculty of Law and Social Sciences',
    6: 'Faculty of Humanities'
}

# Assign a faculty to each degree
faculty_degrees = {
    1: [1, 2, 3, 4],            # Faculty of Computer Science and Engineering
    2: [5, 6, 7, 8],            # Faculty of Natural Sciences
    3: [9],                     # Faculty of Medicine and Health Sciences
    4: [10, 11, 13, 14],        # Faculty of Economics and Business
    5: [12, 15, 19, 20, 21],    # Faculty of Law and Social Sciences
    6: [16, 17, 18, 22]         # Faculty of Humanities
}

# Create the faculty dataframe
faculty = pd.DataFrame({
    'facultyID': list(faculties.keys()),
    'facultyName': list(faculties.values()),
    'universityID': university_id * len(faculties)
})

## 'Degree' table
# Decide, for each student, if they're doing undergraduate or postgraduate.
dic_degree_category = {student: np.random.choice(['Undergraduate', '
   Postgraduate']) for student in student_id}

# Get the facultyID for each degree that is taught
faculty_ids = []
for degree_id in degree_dict.values():
    for faculty_id, grados in faculty_degrees.items():
        if degree_id in grados:
            faculty_ids.append(faculty_id)
            break

# Create the degree dataframe
grado = pd.DataFrame({
    'degreeID': list(degree_dict.values()),
    'degreeName': list(degree_dict.keys()),
    'facultyID': faculty_ids[:len(degree_dict)],
    # Decide whether a certain degree belongs to an undergraduate or
   postgraduate program
    'category': [np.random.choice(['Undergraduate', 'Postgraduate']) for _ in
   degree_dict]
})

## 'Module' table
# List of template string for module names
module_templates = ["Introduction to {degree}",
                    "{degree} Fundamentals",
                    "Advanced Topics in {degree}",
                    "{degree} Project",
                    "Research Methods in {degree}",
                    "Ethics and Professional Practice in {degree}"]

module_dict = {}
module_id = 1
# Iterate through each degree
for degree in degrees:
    # Iterate through the list of module templates
    for template in module_templates:
        module_name = template.format(degree = degree) # Replace '{degree}' for
    the actual degree name
        module_dict[module_id] = module_name # Assign an ID to the module name
        module_id += 1

# Assign each degreeID its corresponding moduleIDs
degree_module = {}
module_id = 1
# Iterate through the degreeIDs
for deg_id in range(1, len(degrees) + 1):
    modules_for_degree = [] # Saves the moduleIDs for the current degree
    # Iterate through the number of modules
    for _ in range(len(module_templates)):
        modules_for_degree.append(module_id) # Assigns the moduleID to the
   current degree
        module_id += 1
    # Associate the current degreeID with its corresponding moduleIDs
    degree_module[deg_id] = modules_for_degree

## 'Professor' table
professor_name = ['Emma', 'Liam', 'Olivia', 'Noah', 'Ava', 'Elijah', 'Sophia',
   'James', 'Amelia', 'Benjamin', 'Mia', 'Charlotte', 'Henry', 'Isabella', '
   Ethan', 'Harper', 'Alexander', 'Evelyn', 'Sebastian', 'Gabriel']

professor_surname = ['Williams', 'Taylor', 'Anderson', 'Clark', 'Harris', '
   Baker', 'Mitchell', 'Carter', 'Evans', 'Parker', 'Scott', 'Reed', 'Morris',
   'Ward', 'Watson', 'Brooks', 'Murphy', 'Bell', 'Cole', 'Bennett']
```

```
# The professor's email will be their name + surname + '@gmail.com'
professor_email = []
for i in range(len(professor_name)):
    professor_email.append(str(professor_name[i]).lower() + str(
   professor_surname[i]).lower() + '@gmail.com')

# Generate IDs for the professors
professor_id = np.arange(1, len(professor_name) + 1)

# Generate UK number phones
professor_phone = []
for i in range(len(professor_id)):
    if random.random() < 0.3:  # 30% chances of the phone number being null
        professor_phone.append(None)
    else:
        professor_phone.append('+44 7********' + str(random.randint(00, 99)))

## 'Department' table
departments = ['Computing and Software Engineering', 'Data and AI', 'Mechanical
    and Electrical Engineering', 'Mathematics and Statistics', 'Natural
   Sciences', 'Health and Medicine', 'Economics and Finance', 'Business and
   Marketing', 'Law and Criminology', 'Social and Behavioral Sciences', '
   Humanities and Philosophy', 'Politics and International Studies']

# Assign an ID to each department
department_dict = {department: i + 1 for i, department in enumerate(departments
   )}

# Assign the modules to the corresponding departments
dict_department_modules = {
    1: [1, 2, 3, 4, 5, 6],
    2: [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18],
    3: [19, 20, 21, 22, 23, 24],
    4: [25, 26, 27, 28, 29, 30],
    5: [31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,
   48],
    6: [49, 50, 51, 52, 53, 54],
    7: [55, 56, 57, 58, 59, 60, 79, 80, 81, 82, 83, 84],
    8: [61, 62, 63, 64, 65, 66, 85, 86, 87, 88, 89, 90],
    9: [67, 68, 69, 70, 71, 72, 115, 116, 117, 118, 119, 120],
    10: [73, 74, 75, 76, 77, 78, 109, 110, 111, 112, 113, 114],
    11: [91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106,
    107, 108],
    12: [121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132]
}

# Randomly assign each professor to a unique department
dept_ids = list(range(1, len(departments) + 1))
prof_ids = list(range(1, len(professor_id) + 1))
random.shuffle(prof_ids)

dic_professor_department = {}
for i in range(len(dept_ids)):
    dic_professor_department[prof_ids[i]] = dept_ids[i]
for i in range(len(dept_ids), len(prof_ids)):
    dic_professor_department[prof_ids[i]] = random.choice(dept_ids)

# Assign modules to professors(same department)
dic_module_professor = {}
for mod_id in range(1, 133):
    department_id = next(dept for dept, mods in dict_department_modules.items()
    if mod_id in mods)
    # List of professors that belong to the same department
    eligible_professors = [p_id for p_id, dept_id in dic_professor_department.
   items() if dept_id == department_id]
    # Randomly assign one of the professors to the module
    dic_module_professor[mod_id] = random.choice(eligible_professors)

## 'Module' table
degree_ids = []
department_ids = []
for mod_id in module_dict.keys():
    # Find the degree and department for each module
    deg_id = next((d_id for d_id, mods in degree_module.items() if mod_id in
   mods), None)
    dep_id = next((dpt_id for dpt_id, mods in dict_department_modules.items()
   if mod_id in mods), None)
    degree_ids.append(deg_id)
    department_ids.append(dep_id)

professor_ids = [dic_module_professor.get(m_id, None) for m_id in module_dict]

# Generate the 'module' dataframe
module = pd.DataFrame({
    'moduleID': list(module_dict.keys()),
    'moduleName': list(module_dict.values()),
    'credits': np.random.randint(6, 12, len(module_dict)),
    'degreeID': degree_ids,
    'professorID': pd.array(professor_ids, dtype = pd.Int64Dtype()),
    'departmentID': department_ids
})

# Generate the 'Professor' dataframe
professor = pd.DataFrame({
    'professorID': professor_id,
    'email': professor_email,
    'name': professor_name,
    'surname': professor_surname,
    'phone': professor_phone,
    'departmentID': [dic_professor_department[p] for p in professor_id],
    'universityID': university_id * len(professor_id)
})

# Assign the departments to each faculty
faculty_department = {
    1: [1, 2, 3],          # Faculty of Computer Science and Engineering
    2: [4, 5],             # Faculty of Natural Sciences
    3: [6],                # Faculty of Medicine and Health Sciences
    4: [7, 8],             # Faculty of Economics and Business
    5: [9, 10],            # Faculty of Law and Social Sciences
    6: [11, 12]            # Faculty of Humanities
```

```
}

# Create the 'Department' dataframe
department = pd.DataFrame({
    'departmentID': list(department_dict.values()),
    'departmentName': list(department_dict.keys()),
    # Get the facultyID by iterating over each departmentID (values of
   department_dict)
    'facultyID': [next((fid for fid, depts in faculty_department.items() if
   dept_id in depts), None) for dept_id in department_dict.values()]
})

## 'Enrollment' table
enrollment = pd.DataFrame(columns = ['term', 'year', 'grade', 'studentID', '
   moduleID'])
enrollment_rows = []

# Iterate over each row in the 'student' DataFrame
for _, row in student.iterrows():
    st_id = row['studentID'] # Get the studentID from the current row
    deg_cat = dic_degree_category.get(st_id, 'Undergraduate').lower() # Get the
    degree category of the student ('Undergraduate' if the ID is not found)
    fdate = row['finishDate'] # Get the finish date of the student
    module_ids = module['moduleID'].tolist() # Get a list of moduleIDs

    # If the student has completed their studies
    if pd.notnull(fdate):
        # Undergraduate students
        if deg_cat == 'undergraduate':
            # Assign module marks for the first two years (12 modules, 6 per
   year, 3 per term).
            for i, m_id in enumerate(module_ids[:12]):
                year = 1 if i < 6 else 2   # Ordinal Data
                term = 1 if i % 6 < 3 else 2
                grade = np.random.uniform(50, 100) # Randomly assign a grade in
    between 50 and 100.
                enrollment_rows.append({
                    'term': term, 'year': year, 'grade': round(grade, 2),
                    'studentID': st_id, 'moduleID': m_id
                })
        # Postgraduate students
        elif deg_cat == 'postgraduate':
            # Assign module marks for one year (12 modules, 6 per term).
            for i, m_id in enumerate(module_ids[:12]):
                year = 1
                term = 1 if i < 6 else 2
                grade = np.random.uniform(50, 100)
                enrollment_rows.append({
                    'term': term, 'year': year, 'grade': round(grade, 2),
                    'studentID': st_id, 'moduleID': m_id
                })

enrollment = pd.DataFrame(enrollment_rows)

# Merge to get the startDate of each student
enrollment = enrollment.merge(
    student[['studentID','startDate']],
    on = 'studentID',
    how = 'left')

# Calculate the year in which they are studying this 'year'
enrollment['enrollmentYear'] = enrollment['startDate'].dt.year + (enrollment['
   year'] - 1)

# Defining the cutoff year
current_year = pd.Timestamp.now().year

# The enrollmentYear cannot be greater than the current year
enrollment.loc[enrollment['enrollmentYear'] > current_year, 'grade'] = None

# Remove auxiliar columns
enrollment.drop(columns = ['startDate', 'enrollmentYear'], inplace = True)

# Save the dataframes to CSV files
university.to_csv('data/university_csv.csv', index = False)
student.to_csv('data/student_csv.csv', index = False)
faculty.to_csv('data/faculty_csv.csv', index = False)
grado.to_csv('data/degree_csv.csv', index = False)
module.to_csv('data/module_csv.csv', index = False)
professor.to_csv('data/professor_csv.csv', index = False)
department.to_csv('data/department_csv.csv', index = False)
enrollment.to_csv('data/enrollment_csv.csv', index = False)
```

B SQL code to create the database

In DB Browser (SQLite), we first need to import the CSVs containing the data generated previously in the Python code. After that, we can run commands shown below. Once this is done, our data should have been correctly imported into the database.

```
CREATE TABLE University (
     universityID   INTEGER PRIMARY KEY,
     universityName TEXT NOT NULL,
   country        TEXT,
     county         TEXT,
     city           TEXT,
     postcode       TEXT,
     foundationYear INTEGER ,
     universityAge  INTEGER
);

CREATE TABLE Faculty (
     facultyID                INTEGER PRIMARY KEY,
     facultyName              TEXT NOT NULL,
     universityID             INTEGER NOT NULL,
     FOREIGN KEY (universityID) REFERENCES University(universityID)
);

CREATE TABLE Degree (
     degreeID          INTEGER PRIMARY KEY,
     degreeName        TEXT NOT NULL,
     facultyID         INTEGER NOT NULL,
     category          TEXT,
     FOREIGN KEY (facultyID) REFERENCES Faculty(facultyID)
);

CREATE TABLE Department (
     departmentID      INTEGER PRIMARY KEY,
     departmentName    TEXT NOT NULL,
     facultyID         INTEGER NOT NULL,
     FOREIGN KEY (facultyID) REFERENCES Faculty(facultyID)
);

CREATE TABLE Professor (
     professorID              INTEGER  NOT NULL PRIMARY KEY,
     email                    TEXT  NOT NULL UNIQUE,
     name                     TEXT  NOT NULL,
     surname                  TEXT  NOT NULL,
     phone                    TEXT,
     departmentID             INTEGER  NOT NULL,
     universityID             INTEGER  NOT NULL,
   FOREIGN KEY (departmentID) REFERENCES Department(departmentID),
   FOREIGN KEY (universityID) REFERENCES University(universityID)
);

CREATE TABLE Student (
     studentID                INTEGER  NOT NULL PRIMARY KEY,
     email                    TEXT  NOT NULL UNIQUE,
     name                     TEXT  NOT NULL,
     surname                  TEXT  NOT NULL,
     gender                  TEXT,
     nationality             TEXT,
     startDate               DATE,
     finishDate              DATE,
     degreeID                INTEGER  NOT NULL,
     universityID            INTEGER  NOT NULL,
   FOREIGN KEY (degreeID) REFERENCES Degree(degreeID),
   FOREIGN KEY (universityID) REFERENCES University(universityID)
);

CREATE TABLE Module (
     moduleID                INTEGER PRIMARY KEY,
     moduleName              TEXT NOT NULL,
     credits                 INTEGER,
   degreeID                INTEGER  NOT NULL,
     professorID             INTEGER  NOT NULL,
     departmentID            INTEGER NOT NULL,
   FOREIGN KEY (degreeID) REFERENCES Degree(degreeID)
   FOREIGN KEY (professorID) REFERENCES Professor(professorID)
     FOREIGN KEY (departmentID) REFERENCES Department(departmentID)
);

CREATE TABLE Enrollment (
    term              TEXT,
    year              TEXT,
    grade             REAL,
    studentID         INTEGER NOT NULL,
    moduleID          INTEGER NOT NULL,
    PRIMARY KEY (studentID, moduleID),
    FOREIGN KEY (studentID) REFERENCES Student(studentID),
    FOREIGN KEY (moduleID) REFERENCES Module(moduleID)
);

-- foreign kys available
PRAGMA foreign_keys = ON;


-- INSERTING INTO TABLES
INSERT INTO University (universityID, universityName, country, county, city,
   postcode, foundationYear, universityAge)
SELECT universityID, universityName, country, county, city, postcode,
   foundationYear, universityAge
FROM university_csv;

INSERT INTO Faculty (facultyID, facultyName, universityID)
SELECT facultyID, facultyName, universityID
FROM faculty_csv;

INSERT INTO Degree (degreeID, degreeName, facultyID, category)
SELECT degreeID, degreeName, facultyID, category
FROM degree_csv;

INSERT INTO Department (departmentID, departmentName, facultyID)
SELECT departmentID, departmentName, facultyID
FROM department_csv;
```

```sql
INSERT INTO Professor (professorID, email, name, surname, phone, departmentID,
   universityID)
SELECT professorID, email, name, surname, phone, departmentID, universityID
FROM professor_csv;

INSERT INTO Student (studentID, email, name, surname, gender, nationality,
   startDate, finishDate, degreeID, universityID)
SELECT studentID, email, name, surname, gender, nationality, startDate,
   finishDate, degreeID, universityID
FROM student_csv;

INSERT INTO Module (moduleID, moduleName, credits, degreeID, professorID,
   departmentID)
SELECT moduleID, moduleName, credits, degreeID, professorID, departmentID
FROM module_csv;

INSERT INTO Enrollment (term, year, grade, studentID, moduleID)
SELECT term, year, grade, studentID, moduleID
FROM enrollment_csv;


-- DROP CSVs
DROP TABLE degree_csv;
DROP TABLE department_csv;
DROP TABLE enrollment_csv;
DROP TABLE faculty_csv;
DROP TABLE module_csv;
DROP TABLE professor_csv;
DROP TABLE student_csv;
DROP TABLE university_csv;
```