Question 1 (20 marks)

Description: Database Models with SQLAlchemy

Part a [8 marks]

Consider code given on the first page of 'flight management system' that defines the schema for a simplified system for flight planning and booking. It consists of four main tables:

`Airports:` Contains information about airports, including their unique code, name, city, and country.

`Aircrafts:` Stores details about aircraft, such as registration number, model, and seating capacity

`Flights:` Represents individual flights with departure and arrival times, associated aircraft, and departure and arrival airports.

`Bookings:` Records passenger bookings for specific flights, including passenger names and seat numbers.

Your task:

Draw all the four tables by filling some random values for each attribute such that all the relationships within and between the table definitions are covered in the example tables. Create at least two rows for each table. DO NOT write the code for inserting rows.

Assumptions: The code for table creation is complete and syntax is correct. The time format is 'YYYY-MM-DD HH:MM:SS'.

Airport

id	code	name	city	country
1	JFK	John F. Kennedy International Airport	New York	USA
2	LAX	Los Angeles International Airport	Los Angeles	USA
3	YYZ	Toronto International Airport	Toronto	CAN

Aircraft

id	registration	model	capacity
1	N12345	Boeing 737	150
2	N67890	Airbus A320	180

Flight

each airport can have multiple arrival and departures(also these airports must exist in the Airport table)
each aircraft can be used for multiple flights(aircraft must exist in aircraft table)

id	departure_time	arrival_time	aircraft_id	departure_airport_id	arrival_airport_id
1	2024-04-16 8:00:00	2024-04-16 11:00:00	1	1	2
2	2024-04-16 9:00:00	2024-04-16 12:00:00	2	2	1
3	2024-05-16 9:00:00	2024-04-16 11:00:00	1	1	3

Booking

Each flight can have multiple bookings(flight id must exist in Flight table)

id	flight_id	passenger_name	seat_number
1	1	Alice	A1
2	1	Bob	B1
3	2	Charlie	C1

Part b [6 marks]

Identify and describe all ***the relationships between the tables***.

The relationships between the tables can be *many-to-many, many-to-one, one-to-many or one-to-one*. You may describe the relationships briefly using **foreign key and/or backref**. Use the examples from the sample tables created in part a for explanation.

Airports to Flights Relationship:

- Foreign Key: In the Flights table, the departure_airport_id and arrival_airport_id columns serve as foreign keys referencing the primary key (id) of the Airports table.
- Backref: In the Airports table, the relationship with flights departing from the airport is represented by the departures attribute, and the relationship with flights arriving at the airport is represented by the arrivals attribute.
- Description:
    - one-to-many: Each airport can have multiple flights departing from it and multiple flights arriving at it.
    - one-to-one: Each flight is associated with one departure airport and one arrival airport.
- Example: Flight 1 departs from John F. Kennedy International Airport (JFK) and arrives at Los Angeles International Airport (LAX).

Aircrafts to Flights Relationship:

- Foreign Key: In the Flights table, the aircraft_id column serves as a foreign key referencing the primary key (id) of the Aircrafts table.
- Backref: In the Aircrafts table, the relationship with flights operated by the aircraft is represented by the flights attribute.
- Description:
    - one-to-many: Each aircraft can operate multiple flights.
    - one-to-one: Each flight is operated by one aircraft.
- Example: Flight 1 is operated by a Boeing 737 (registration N12345).

Flights to Bookings Relationship:

- Foreign Key: In the Bookings table, the flight_id column serves as a foreign key referencing the primary key (id) of the Flights table.
- Backref: In the Flights table, the relationship with bookings associated with the flight is represented by the bookings attribute.
- Description:
    - one-to-many: Each flight can have multiple bookings associated with it.
    - one-to-one: Each booking is associated with one flight.
- Example: Booking 1 is for Flight 1.

Part c [6 marks]

Suppose you want to enhance the existing flight booking system by adding a table to store information about passengers' luggage. Design a table named `Luggage` to store this information and establish relationship between the `Flights` table and the `Luggage` table.

What attributes would you include in the `Luggage` table, and how would you establish this relationship?

To add a table for storing information about passengers' luggage, we can include attributes such as id (primary key), passenger_name, weight, description, and flight_id (foreign key referencing the Flights table). This will establish a one-to-many relationship between flights and luggage, as each flight may have multiple pieces of luggage associated with it.

For example:

```
class Luggage(db.Model):
  __tablename__ = 'luggage'

  id = db.Column(db.Integer, primary_key=True)
  passenger_name = db.Column(db.String, nullable=False)
  weight = db.Column(db.Float, nullable=False)
  description = db.Column(db.String)
  flight_id = db.Column(db.Integer, db.ForeignKey('flights.id'), nullable=False)

  flight = db.relationship('Flight', backref='luggage')

  def __repr__(self):
    return f'<Luggage {self.passenger_name}>'
```

In this design, the Luggage table stores information about passengers' luggage, including the passenger's name, weight of the luggage, a description (optional), and the associated flight ID. The flight_id attribute establishes a foreign key relationship with the Flights table, creating a one-to-many relationship between flights and luggage.

With this setup, each flight can have multiple pieces of luggage associated with it, allowing for efficient tracking of passengers' luggage within the flight booking system.

Question 2 (15 marks)

Description: Web Dev and Databases – Flask and SQLite

Complete the `search_flights` method in Python Flask.
This method lets the user search for flights using `departure_airport_id`, `arrival_airport_id`, and `aircraft_id`. This information is obtained through a page `'search_flights.html'`, which stores a form to get this information from user. Based on this search, `'flight_results.html'` page is rendered, which will display the result of user's search by getting the result from database.
Your task:

1. Complete the following app configurations on this page.
2. Complete the `search_flights` method in Flask application (next page). Create and store each criterion for searching flights in a separate session variable. If all search criteria are found in session, then retrieve the search criteria from session variables and display the flight details for that search.
   If the search is not in session, then display `search_flight.html` page.

Note: Assume that all the imports are complete. You may assume that the `'search_flights.html'` and `'flight_results.html'` templates exist, and you do not need to create them for this question. Also, work only with the `'flights'` table to retrieve information about existing flights. Assume that for every search, there is a flight in the database. You may define helper functions.

```
from flask import Flask, request, render_template, session, redirect, url_for,
flash
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///flights.db'
app.config['SECRET_KEY']='8a0f946f1471e113e528d927220ad977ed8b2cce63303beff1'
db = SQLAlchemy(app)

@app.route('/search_flights', methods=['GET', 'POST'])
def search_flights():
    if request.method == 'GET':
        if 'departure_airport_id' in session and 'arrival_airport_id' in
            session and 'aircraft_id' in session:
            flights = flight_query(session['departure_airport_id'],
                                   session['arrival_airport_id'],
                                   session['aircraft_id'])
            return render_template('flight_results.html', flights=flights)
        else:
            return render_template('search_flights.html')
    else:
        # Handle form submission
        departure_airport_id = request.form['departure_airport_id']
        arrival_airport_id = request.form['arrival_airport_id']
        aircraft_id = request.form['aircraft_id']

        # Perform flight search based on criteria
        flights = flight_query(departure_airport_id, arrival_airport_id,
                                   aircraft_id)
```

```
        # Store the search criteria in session
        session['departure_airport_id'] = departure_airport_id
        session['arrival_airport_id'] = arrival_airport_id
        session['aircraft_id'] = aircraft_id

        return render_template('flight_results.html', flights=flights)


def flight_query(departure_airport_id, arrival_airport_id, aircraft_id):
    flights = Flight.query.filter_by(
            departure_airport_id=departure_airport_id,
            arrival_airport_id=arrival_airport_id,
            aircraft_id=aircraft_id
        ).all()
    return flights
```

Question 3 (5 marks)

Suppose `'flight_results.html'` from question 2 displays the result of user's flight search using the table display in html. Assume that the information about the relevant flights is received by the html page through a variable named `flight`. Complete following html code using jinja2 template for `'flight_results.html'`.

```
<body>
    <h1>Flight Search Results</h1>
    <table>
        <thead>
            <tr>
                <th>Flight ID</th>
                <th>Departure Airport</th>
                <th>Arrival Airport</th>
                <th>Departure Time</th>
                <th>Arrival Time</th>
            </tr>
        </thead>
        <tbody>
            {% for flights in flights %}
            <tr>
                <td>{{ flights.id }}</td>
                <td>{{ flights.departure_airport_id }}</td>
                <td>{{ flights.arrival_airport_id }}</td>
                <td>{{ flights.departure_time }}</td>
                <td>{{ flights.arrival_time }}</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
</body>
</html>
```

Question 4 (15 marks)

Part a [5 marks]

You have a simple HTML document containing a class container with some text, button, and script tag. Your task is to create a CSS ***media query that creates a responsive grid layout for class `'container'`*** based on screen size.

```
<body>
    <div class="container">
        <div class="item">Item 1</div>
        <div class="item">Item 2</div>
        <div class="item">Item 3</div>
        <div class="item">Item 4</div>
        <div class="item">Item 5</div>
        <div class="item">Item 6</div>
        <div class="item">Item 7</div>
        <div class="item">Item 8</div>
        <div class="item">Item 9</div>
    </div>
    <button id="colorButton">Change Background Color</button>
    <script src="script.js"></script>
</body>
```

The layout should display *three columns on large screens (more than 1200px), two columns on medium screens (between 768 and 1200 px), and a single column on small screens (smaller than 768 px)*. Additionally, each column should have *equal width*. Media query should be written only for class `container`. DO NOT write code for `button` or `script` tags.

Complete the following code and write media query. Assume HTML code and script.js code are complete and run without errors.

```
.container {
  display: grid;
  /* 1fr is for 1 part of the available space */
  grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
  grid-gap: 20px;
}

/* Ensure each column has equal width and content is centered */
.item {
  width: 100%;
  text-align: center;
}

/* Media queries for responsiveness */
@media screen and (max-width: 1200px) {
  .container {
    grid-template-columns: 1fr 1fr;
  }
}

@media screen and (max-width: 768px) {
  .container {
    grid-template-columns: 1fr;
  }
}
```

Part b [5 marks]

Write JavaScript code to select the button element from the HTML document on previous page.
Attach an event listener to the button element to listen for click events.
When the button is clicked, execute a function that changes the background color of the document's body to a random color.

Note: Use following variable to generate random color

```
randomColor = `rgb(${Math.floor(Math.random() * 256)},
                  ${Math.floor(Math.random() * 256)},
                  ${Math.floor(Math.random() * 256)})`;
```

```
// Select the button element
const colorButton = document.getElementById('colorButton');

// Add event listener to the button
colorButton.addEventListener('click', function() {
    // Generate random RGB values for background color
    const randomColor = `rgb(${Math.floor(Math.random() * 256)},
${Math.floor(Math.random() * 256)}, ${Math.floor(Math.random() * 256)})`;

    // Change background color of the document's body
    document.body.style.backgroundColor = randomColor;
});
```

Part c [5 marks]

Briefly explain the concept and importance of DOM in web applications both front and back end.

In web development, the Document Object Model (DOM) is a programming interface that represents the structure of HTML documents as a hierarchical tree-like structure. Each node in the DOM tree represents an element, attribute, or text within the document, allowing developers to manipulate and interact with the document dynamically.
Front-end web development:

Front-end developers use the DOM extensively to create dynamic and interactive web applications. They can manipulate DOM elements using JavaScript to update the content, structure, and style of web pages dynamically based on user interactions or data changes.

The DOM also provides mechanisms for handling user events such as clicks, mouse movements, and keyboard inputs. Front-end developers can attach event listeners to DOM elements to respond to these events and trigger specific actions, enabling interactive features like form validations, dropdown menus, and sliders.

Back-end web development:

In frameworks like Jinja2, developers use templating engines to generate HTML content dynamically on the server. Jinja2 templates allow developers to embed logic, iterate over data collections, and conditionally render content, resulting in dynamic web pages tailored to specific user requests.

Data Transformation: Back-end developers may need to transform data from various sources (e.g., databases, APIs) into HTML content to be rendered in web pages. They can use Jinja2 templates to format and structure data dynamically, ensuring that the resulting HTML documents meet the requirements of the front-end design

Question 5 (20 marks)

Description: (RA and SQL queries)

Note*: Following schema is different than one used in Question 1, Question 2, and Question 3*. Consider following schema that keeps track of **airline flight information**. This schema is designed to manage airline flight information, including details about flights, aircraft, certifications, and employees.

```
Flights(flno: integer, from: string, to: string, distance: integer,
          departs: time, arrives: time, price: real)
Aircraft(aid: integer, aname: string, cruisingrange: integer)
Certified(eid: integer, aid: integer)
Employees(eid: integer, ename: string, salary: integer)
```

- `cruisingrange` attribute defines Maximum distance the aircraft can travel without refueling.
- The `Employees` relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft, and only pilots are certified to fly.
- The `Certified` table establishes a many-to-many relationship between employees and aircraft, indicating which employees are certified to operate which aircraft.

Part a [5 marks]

Find the `aids` of all aircraft that can be used on routes from Los Angeles to Chicago.

```
SELECT A.aid
FROM   Aircraft A
WHERE  A.cruisingrange > ( SELECT MIN (F.distance)
                           FROM   Flights F
                           WHERE  F.from = 'Los Angeles' AND F.to = 'Chicago' )
```

Part b [10 marks]
Find the names of aircraft such that all pilots certified to operate them have salaries more than $80,000.

```
SELECT DISTINCT A.aname
FROM    Aircraft A
WHERE   A.Aid IN (SELECT C.aid
                  FROM    Certified C, Employees E
                  WHERE   C.eid = E.eid AND
                  NOT EXISTS ( SELECT *
                               FROM    Employees E1
                               WHERE   E1.eid = E.eid AND E1.salary < 80000 ))
```

Part c [5 marks]
Convert following relational algebra queries into English sentences.

1. $R1 \coloneqq \pi_{eid}(\sigma_{cruisingrange>3000}(Aircraft \bowtie Certified)))$
   $\pi_{enames}(Employees \bowtie (R1 - \pi_{eid}(\sigma_{name='Boeing'}(Aircraft \bowtie Certified))))$

   Find the names of pilots who can operate planes with a range greater than 3,000 miles but are not certified on any Boeing aircraft.

2. $R1 \coloneqq \rho_{E1} Employees$
   $R2 \coloneqq \rho_{E2} Employees$
   $R3 \coloneqq \rho_{E3} \left(\pi_{E2.eid}\left(E1 \bowtie_{E1.salary>E2.salary} E2\right)\right)$
   $R4 \coloneqq \rho_{E4} (E2 \bowtie E3)$
   $R5 \coloneqq \rho_{E5} (E2 \bowtie E3)$
   $R6: = \rho_{E6} (\pi_{E5.eid}\left(E4 \bowtie_{E1.salary>E5.salary} E5\right))$
   $(\pi_{eid} E3) - E6$

   Find the eids of employees who make the second highest salary.

Question 6 (10 marks) Multiple Choice Questions

1. Which of the following is True? Select ALL that apply.
   a) All primary keys are superkeys
   b) All superkeys are primary keys
   c) All candidate keys are primary keys
   d) All candidate keys are superkeys
   e) All superkeys are candidate keys

2. Consider the following schema for an online store:

```
Customers(customer_id: int, name: varchar(255), email: varchar(255))
Orders(order_id: int, customer_id: int, order_date: date)
Products(product_id: int, name: varchar(255), price: decimal)
OrderDetails(order_detail_id: int, order_id: int, product_id: int, quantity: int)
```

what type of relationship exists between the `Products` and `OrderDetails` tables?
   a) One-to-One
   b) One-to-Many
   c) Many-to-One
   d) Many-to-Many
   e) None of the above

3. Which of the following command is correct to delete the values in the relation `flights`?
   a) Delete from flights;
   b) Delete from flights where Id =’Null’;
   c) Remove table flights;
   d) Drop table flights;
   e) None of the above

4. Consider the following HTML structure:

```
<div class="container">
    <div class="intro">
        <p>This is paragraph 1.</p>
        <p>This is paragraph 2.</p>
    </div>
    <p class="intro">
        This is paragraph 3</p>
</div>
```

Which of the following CSS selectors selects only the <p> element containing the text "This is paragraph 3."?
Select ALL that apply.
   a) .intro p
   b) p.intro
   c) .container p
   d) .container .intro p
   e) p > *

5. Which of the following statements about SQLite is true?

a) SQLite is a serverless, self-contained, zero-configuration, transactional SQL database engine.
b) SQLite is primarily designed for large-scale enterprise-level applications.
c) SQLite requires a separate server process to be running in order to access the database.
d) SQLite is not suitable for mobile application development.
e) None of the above

6. Consider the following JavaScript code snippet:

```
var element = document.createElement("button");
element.textContent = "Click Me";
element.setAttribute("id", "myButton");
element.addEventListener("click", function() {
    alert("Button clicked!");
});

document.body.appendChild(element);
```

Which of the following accurately describes what the code snippet does?

a) **Creates a button element with the text "Click Me", sets its id to "myButton", and appends it to the document body.**
b) Attaches a click event listener to the document body that triggers an alert when clicked.
c) Appends a new button element with the text "Click Me" to the end of the document body.
d) Creates a button element with the text "Click Me" and sets its id to "myButton".
e) Adds a click event listener to the button element that triggers an alert when clicked.

7. What is the primary purpose of the secret key in a Flask application?

a) To encrypt sensitive data stored in the database.
b) To authenticate users during login.
c) To establish a secure connection between the client and the server.
d) To sign session cookies and prevent tampering.
e) To define the routing paths for different views in the application.

8. Which of the following statements accurately describes the role of Jinja2 in web development?

a) Jinja2 is a front-end JavaScript library used for handling user interactions and DOM manipulation.
b) Jinja2 is a back-end Python framework used for routing and handling HTTP requests.
c) Jinja2 is a templating engine that enables the dynamic generation of HTML content by embedding Python-like syntax within HTML templates.
d) Jinja2 is a database management system used for storing and querying structured data.
e) Jinja2 is a styling framework used for creating responsive and visually appealing web pages.

9. Which of the following statements accurately describes "localhost" in web development?

a) Localhost refers to a remote server accessible only via the internet.
b) Localhost is a specific IP address that points to the local machine itself.
c) Localhost is a cloud-based hosting service used for deploying web applications.
d) Localhost is a reserved domain name used exclusively for testing purposes.
e) Localhost refers to a virtual environment created by virtualization software for web development.

10. This mark is Free!