Unit 4: Database Integration

4.1 Introduction to Databases

4.2 Overview of relational (SQL) vs. NoSQL databases

4.3 Database Connectivity with Flask

4.3.1 Connecting Flask applications to databases using SQL Alchemy

4.4 Performing CRUD operations using HTML template

4.5 Handling database connections and transactions

Introduction to Databases

A **database** is an organized collection of data stored and accessed electronically. Databases are essential for storing, retrieving, and managing data in web applications.

Key Database Concepts

1. **Schema**
    - Defines the structure (tables, fields, relationships).
    - In SQL: Predefined before inserting data.
    - In NoSQL: Can evolve dynamically.
2. **ACID Properties (for SQL Databases)**
    - **Atomicity:** Transactions are all-or-nothing.
    - **Consistency:** Data remains valid after transactions.
    - **Isolation:** Concurrent transactions don't interfere.
    - **Durability:** Committed data survives crashes.
3. **CAP Theorem (for NoSQL Databases)**

- **Consistency:** All nodes see the same data.
- **Availability:** Every request gets a response.
- **Partition Tolerance:** System works despite network failures.

(NoSQL databases often sacrifice Consistency for Availability.)

Types of Databases:

1. **Relational Databases (SQL):**
    - Store data in structured tables with rows and columns.
    - Use **SQL (Structured Query Language)** for queries.
    - Examples: MySQL, PostgreSQL, SQLite, Oracle.

Structure:

- Data stored in **tables** with rows (records) and columns (fields).
- Relationships defined via **foreign keys**.

Example (PostgreSQL):

```sql
-- Create a table
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    department_id INT REFERENCES departments(id)
);

-- Insert data
INSERT INTO employees (name, department_id) VALUES ('Alice', 1);
```

Pros:

✔️ Strong consistency

✔️ Complex queries (JOINs, subqueries)
✔️ Mature ecosystem

Cons:
✖️ Scaling requires vertical upgrades
✖️ Schema changes can be costly

o

2. **NoSQL Databases:**

o Store unstructured or semi-structured data (JSON, key-value pairs, documents).

o Flexible schema, scalable for large datasets.

o Examples: MongoDB, Cassandra, Redis, Firebase.

Types:

1. **Document Stores (MongoDB)**

o Data stored as JSON-like documents.

```
{
  "_id": 1,
  "name": "Alice",
  "department": "Engineering"
}
```

2. **Key-Value Stores (Redis)**

o Simple `key:value` pairs.

```
redis.set("user:1", "{'name': 'Alice'}")
```

3. **Column-Family (Cassandra)**

o Optimized for large-scale data.

4. **Graph Databases (Neo4j)**

- o Store relationships between entities.

Pros:
✔️ Horizontal scaling
✔️ Flexible schema
✔️ High performance for specific use cases

Cons:
✖️ Limited query capabilities
✖️ No standard query language

4.2 Overview of Relational (SQL) vs. NoSQL Databases

Feature	Relational (SQL)	NoSQL
Structure	Tables (Rows & Columns)	Documents, Key-Value, Graph
Schema	Fixed (Predefined)	Dynamic (Flexible)
Scalability	Vertical (Hardware Upgrade)	Horizontal (Distributed)
Query Language	SQL	Custom (e.g., MongoDB queries)
Use Case	Complex queries, transactions	Big data, real-time apps

Example:

- **SQL (MySQL):**

```sql
CREATE TABLE users (id INT PRIMARY KEY, name VARCHAR(100), email
VARCHAR(100));
INSERT INTO users VALUES (1, 'John Doe', 'john@example.com');
```

- **NoSQL (MongoDB):**

json

```
db.users.insertOne({ id: 1, name: "John Doe", email: "john@example.com" });
```

4.3 Database Connectivity with Flask

Flask can connect to databases using **ORMs (Object-Relational Mappers)** like **SQLAlchemy** or direct drivers (e.g., `psycopg2` for PostgreSQL).

4.3.1 Connecting Flask to Databases using SQLAlchemy

SQLAlchemy is a popular ORM that allows Python to interact with SQL databases.

Steps to Connect Flask with SQLAlchemy:

1. **Install Required Packages:**

```
pip install flask flask-sqlalchemy
```

2. **Configure Database URI in Flask:**

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///mydatabase.db'  # SQLite example
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
```

3. **Define a Model (Table):**

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)
```

4. **Create Tables:**

```python
with app.app_context():
    db.create_all()  # Creates the database tables
```

4.4 Performing CRUD Operations using HTML Templates

CRUD = **Create, Read, Update, Delete**

Example: Flask CRUD with SQLAlchemy

1. Create (Insert)

```python
@app.route('/add', methods=['POST'])
def add_user():
    name = request.form['name']
    email = request.form['email']
    new_user = User(name=name, email=email)
    db.session.add(new_user)
    db.session.commit()
    return redirect('/users')
```

2. Read (Fetch)

```python
@app.route('/users')
def list_users():
    users = User.query.all()  # Fetch all users
    return render_template('users.html', users=users)
```

3. Update

```python
@app.route('/update/<int:id>', methods=['POST'])
def update_user(id):
    user = User.query.get(id)
    user.name = request.form['name']
    user.email = request.form['email']
```

```
    db.session.commit()
    return redirect('/users')
```

4. Delete

```
@app.route('/delete/<int:id>')
def delete_user(id):
    user = User.query.get(id)
    db.session.delete(user)
    db.session.commit()
    return redirect('/users')
```

HTML Template Example (`users.html`)

Run

```
<h1>Users</h1>
<ul>
  {% for user in users %}
    <li>{{ user.name }} - {{ user.email }}</li>
  {% endfor %}
</ul>
```

4.5 Handling Database Connections and Transactions

- **Transactions** ensure data consistency (all operations succeed or fail together).
- **Session Management** in SQLAlchemy:

```
try:
    user = User(name="Alice", email="alice@example.com")
    db.session.add(user)
    db.session.commit()  # Saves changes
except:
    db.session.rollback()  # Reverts on error
finally:
```

```
    db.session.close()  # Close the session
```