CSC 471 project#5
Put it all together: application development

1. [50] **Basic functionality:** your application must support basic database operations such as looking-up, inserting, deleting, and updating. All these must be implemented in your application program and interact with the database through ODBC or JDBC.

For my application development I'm using sql lite for my data base severer and ODBC I'm using python to interact with sql lite server. In my python file I have functions that can search, insert , delete, and update the Employee and Dependent table. Each function will first establish a connection with the sqlite database then perform the sql query depending on the function then store the desire output into a variable to use to open later in my desire temple html file.

For example one of my function "searchEmployee" :

Establish a connection to my sqlite server by connecting to path where my database is stored.

```
@app.route('/searchEmployee', methods=['POST'])
def search():
    query = request.form['query']
    conn = sqlite3.connect('/Users/brandontiong/Documents/Mysql/employeeTest.db')
```

Executing the sql query in this case it searches the Employee by either First name or SSN ID with any character that contain in the first name or SSN column using the "LIKE" query.

```
cur = conn.cursor()
cur.execute("SELECT * FROM employee WHERE Fname LIKE ? OR ssn LIKE ?", ('%'+query+'%', '%'+query+'%'))
rows = cur.fetchall()
cur.close()
conn.close()
```

All the functions that are in my python file are searchEmployee, searchDependent, insertEmployee, insertDependent, deleteEmployee, deleteDependent, updateEmployee, and updateDependent.

```
@app.route('/searchEmployee', methods=['POST'])
def search():
    query = request.form['query']
    conn = sqlite3.connect('/Users/brandontiong/Documents/Mysql/employeeTest.db')
    cur = conn.cursor()
    cur.execute("SELECT * FROM employee WHERE Fname LIKE ? OR ssn LIKE ?", ('%'+query+'%', '%'+query+'%'))
    rows = cur.fetchall()
    cur.close()
    conn.close()
```

```
@app.route('/searchDependent', methods=['POST'])
def searchDependent():
    query = request.form['query']
    conn = sqlite3.connect('/Users/brandontiong/Documents/Mysql/employeeTest.db')
    cur = conn.cursor()
    cur.execute("SELECT * FROM Dependent WHERE EmployeeSSN LIKE ? OR DependentName LIKE ?", ('%'+query+'%', '%'+query+'%'))
    rows = cur.fetchall()
    cur.close()
    conn.close()
```

```python
def insert():
    conn = sqlite3.connect('/Users/brandontiong/Documents/Mysql/employeeTest.db')
    cursor = conn.cursor()
    query = "INSERT INTO Employee (SSN, DOB, Fname, Minit, Lname, Address) VALUES (?, ?, ?, ?, ?, ?)"

    cursor.execute("SELECT SSN FROM Employee")
```

```python
def insertDependent():
    dependent_name = request.form['dependent_name']
    relationship = request.form['relationship']
    employee_ssn = request.form['employee_ssn']

    conn = sqlite3.connect('/Users/brandontiong/Documents/Mysql/employeeTest.db')
    cur = conn.cursor()
    cur.execute("""
    INSERT INTO Dependent (DependentName, Relationship, EmployeeSSN)
    SELECT ?, ?, ?
    WHERE EXISTS (SELECT 1 FROM Employee WHERE SSN = ?)
```

```python
@app.route('/deleteEmployee', methods=['POST'])
def delete():
    conn = sqlite3.connect('/Users/brandontiong/Documents/Mysql/employeeTest.db')
    cursor = conn.cursor()

    ssn = request.form.get('ssn')
```

```python
def deleteDependent():
    dependent_name = request.form['dependent_name']
    employee_ssn = request.form['employee_ssn']

    conn = sqlite3.connect('/Users/brandontiong/Documents/Mysql/employeeTest.db')
    cur = conn.cursor()
    cur.execute("DELETE FROM Dependent WHERE DependentName=? AND EmployeeSSN=?", (dependent_name, employee_ssn))
```

```python
def update():
    conn = sqlite3.connect('/Users/brandontiong/Documents/Mysql/employeeTest.db')
    cursor = conn.cursor()
    old_ssn = request.form.get('old_ssn')
    new_ssn = request.form.get('new_ssn')
    fname = request.form.get('fname')
    minit = request.form.get('minit')
    lname = request.form.get('lname')
    address = request.form.get('address')

    select_query = "SELECT * FROM Employee WHERE SSN = ?"
    cursor.execute(select_query, (old_ssn,))
    employee = cursor.fetchone()
```

```python
def updateDependent():
    dependent_name = request.form.get('dependent_name')
    relationship = request.form.get('relationship')
    employee_ssn = request.form.get('employee_ssn')
    old_dependent_name = request.form.get('old_dependent_name')

    conn = sqlite3.connect('/Users/brandontiong/Documents/Mysql/employeeTest.db')
    cursor = conn.cursor()

    select_query = "SELECT * FROM Dependent WHERE DependentName = ? AND EmployeeSSN = ?"
    cursor.execute(select_query, (old_dependent_name, employee_ssn))
    dependent = cursor.fetchone()
```

2. [20] **Web based:** users can access your database on line from a web browser. You can implement it using PHP/WAMP/MAMP/LAMP (or Java servlets, MS SQL server, etc.).

For the user to accessed database on line from a web browser I used Flask a imported library of python. Flask is a web framework library that is used to build web applicant that has key functionally such as URL routing, request and response handing, and templates to create html pages.

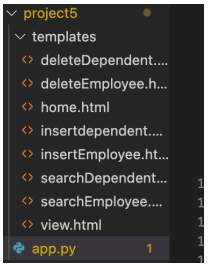
Importing Flask:

```python
from flask import Flask, render_template, request, flash,redirect, url_for
import sqlite3

app = Flask(__name__)
app.secret_key = 'some_secret_key'
@app.route('/')
def home():
    return render_template('home.html')

@app.route('/redirect_to_home')
def redirect_to_home():
    return redirect(url_for('home'))
```

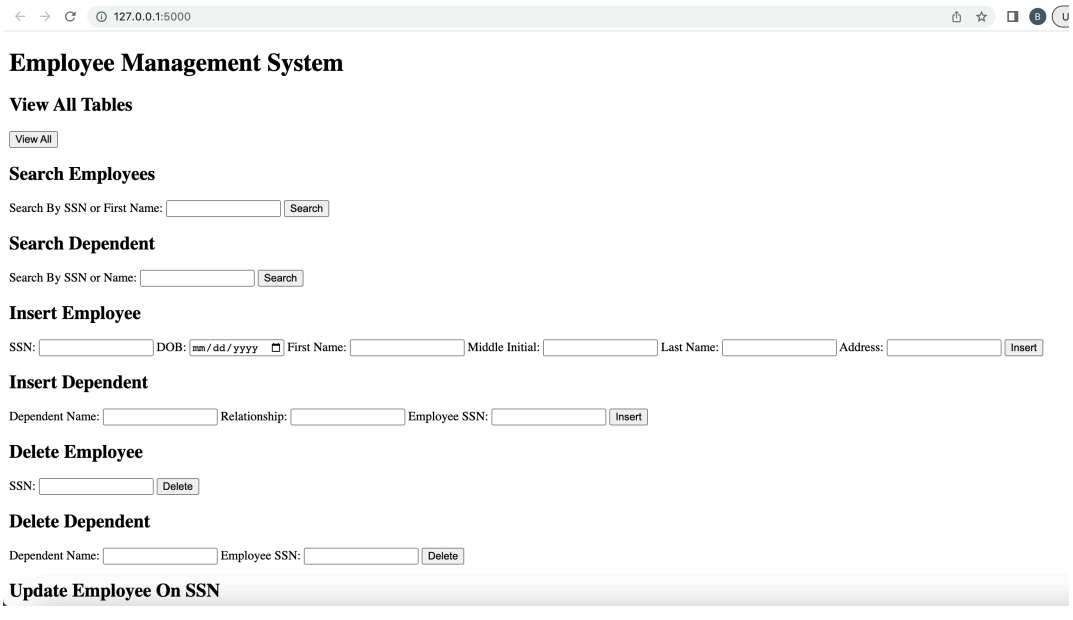
Creating template html pages:



The port to accessed my web browser:

```
(base) brandontiong@brandons-MacBook-Air-2 flask_python % python -u "/Users/brandontiong/Documents/flask_python/project5/app.py"
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

My home web page for my user interactive Employee management System :



3. [20] **Error checking:** your application should be robust enough against bad inputs. For example, a ssn must be exactly 9 digits etc. Is the checking done by application or by database? Explain the trade-offs and give examples of by both.

My Error Checking is mostly done by the application versus the database. The trade off are on the application level error checking the error message can be define by the coder making it clearer error message and easier to debug versus at the database level. But doing error checking at the applicant level is less efficient as it is more time consuming to implement and takes more resource from the application.

Error checking application level:

```
@app.route('/insertEmployee', methods=['POST'])
def insert():
    conn = sqlite3.connect('/Users/brandontiong/Documents/Mysql/employeeTest.db')
    cursor = conn.cursor()
    query = "INSERT INTO Employee (SSN, DOB, Fname, Minit, Lname, Address) VALUES (?, ?, ?, ?, ?, ?)"
    cursor.execute("SELECT SSN FROM Employee")
    existing_ssns = [str(row[0]) for row in cursor.fetchall()]
    ssn = request.form.get('ssn')
    dob = request.form.get('dob')
    fname = request.form.get('fname')
    minit = request.form.get('minit')
    lname = request.form.get('lname')
    address = request.form.get('address')
    if ssn in existing_ssns:
        return f"SSN {ssn} already exists in the database"
    if not ssn.isdigit() or not len(ssn) == 9:
        return "SSN must be number and 9 digits"

    cursor.execute(query, (ssn, dob, fname, minit, lname, address))
```

When inserting a employee into the Employee table my code check if the employee ssn already exist and if the user input for ssn is in correct format of “9 integers “. If user input doesn’t passed my constraints it send the error message of my choosing to user making it easier to understand what they did wrong.

For example the user input for employee SSN already existing in the table:

Insert Employee

SSN: 222334444 DOB: 04/12/2023 First Name: brandon Middle Initial: b.t Last Name: tiong Address: 1224 jamestowne dr Insert

Application level error checking:

127.0.0.1:5000/insertEmployee

SSN 222334444 already exists in the database

In comparison to data level error checking due to ssn being primary key in Employee table:

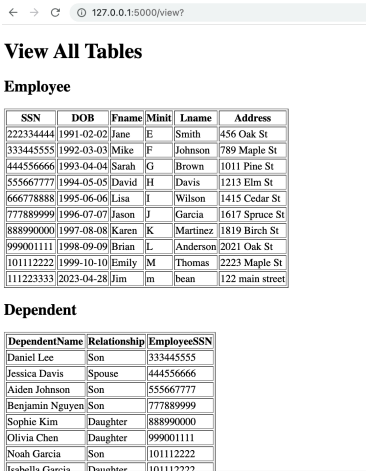
Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

Not a clear error message hard for user to understand what went wrong.

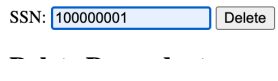
4. [10] **Referential integrity constraint:** let the user see the available input values at runtime (e.g. an existing ssn) during insertion and cascade deletes.

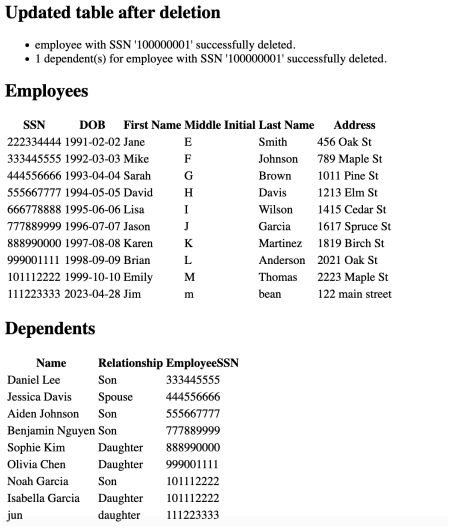
User can see available input values at runtime by hitting view all table to see existing values in each table.



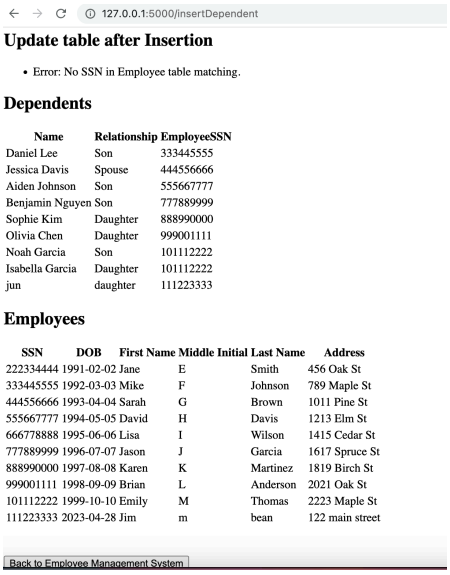
The application enforce referential constraint from the Employee table to the Dependent table since Dependent table ssn is foreign key to ssn of Employee table. When deleting value from the Employee table it will automatically delete from Dependent table if it has same matching ssn ,which is also known as cascade delete. When inserting new values into the Dependent table it has a Error checking if there not a matching ssn in the Employee table. If there isn't then value will not be inserted into the table. When updating in the Employee table if the ssn is change and that old ssn has a corresponding value in the Dependent table then the value in the Dependent table will be updated too. With these referential constraint enforced it make sure the data in the database stay consistent and ensured the relationship between the table are still intact when performing deletion, insertion, and updating.

Delete on cascade:





Insertion referential integrity constraint:



127.0.0.1:5000/insertDependent

Update table after Insertion

- Error: No SSN in Employee table matching.

Dependents

Name	Relationship	EmployeeSSN
Daniel Lee	Son	333445555
Jessica Davis	Spouse	444556666
Aiden Johnson	Son	555667777
Benjamin Nguyen	Son	777889999
Sophie Kim	Daughter	888990000
Olivia Chen	Daughter	999001111
Noah Garcia	Son	101112222
Isabella Garcia	Daughter	101112222
jun	daughter	111223333

Employees

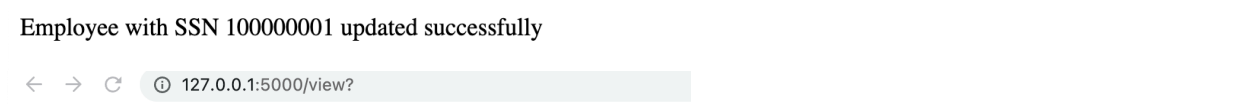
SSN	DOB	First Name	Middle Initial	Last Name	Address
222334444	1991-02-02	Jane	E	Smith	456 Oak St
333445555	1992-03-03	Mike	F	Johnson	789 Maple St
444556666	1993-04-04	Sarah	G	Brown	1011 Pine St
555667777	1994-05-05	David	H	Davis	1213 Elm St
666778888	1995-06-06	Lisa	I	Wilson	1415 Cedar St
777889999	1996-07-07	Jason	J	Garcia	1617 Spruce St
888990000	1997-08-08	Karen	K	Martinez	1819 Birch St
999001111	1998-09-09	Brian	L	Anderson	2021 Oak St
101112222	1999-10-10	Emily	M	Thomas	2223 Maple St
111223333	2023-04-28	Jim	m	bean	122 main street

Back to Employee Management System

Update referential integrity constraint:

Update Employee On SSN

Old SSN: 100000001 New SSN: 100000002 First Name: brandon Middle Initial: b.t Last Name: tiong Address: 1224 jamestowne dr Update



Employee with SSN 100000001 updated successfully

127.0.0.1:5000/view?

Employee

SSN	DOB	Fname	Minit	Lname	Address
222334444	1991-02-02	Jane	E	Smith	456 Oak St
333445555	1992-03-03	Mike	F	Johnson	789 Maple St
444556666	1993-04-04	Sarah	G	Brown	1011 Pine St
555667777	1994-05-05	David	H	Davis	1213 Elm St
666778888	1995-06-06	Lisa	I	Wilson	1415 Cedar St
777889999	1996-07-07	Jason	J	Garcia	1617 Spruce St
888990000	1997-08-08	Karen	K	Martinez	1819 Birch St
999001111	1998-09-09	Brian	L	Anderson	2021 Oak St
101112222	1999-10-10	Emily	M	Thomas	2223 Maple St
111223333	2023-04-28	Jim	m	bean	122 main street
100000002		brandon	b.t	tiong	1224 jamestowne dr

Dependent

DependentName	Relationship	EmployeeSSN
Daniel Lee	Son	333445555
Jessica Davis	Spouse	444556666
Aiden Johnson	Son	555667777
Benjamin Nguyen	Son	777889999
Sophie Kim	Daughter	888990000
Olivia Chen	Daughter	999001111
Noah Garcia	Son	101112222
Isabella Garcia	Daughter	101112222
jun	daughter	111223333
bobbbby	daughter	100000002