SQLite3 Assignments Report

Assignment 1: Creating and Connecting to Database

```
import sqlite3

# Function to create a new SQLite3 database named 'test.db'.
def create_database():
    conn = sqlite3.connect('test.db')
    conn.close()
    print('Database created and successfully connected.')

# Test the function
create_database()
```

This function creates a new SQLite3 database named 'test.db' and connects to it.

```
# Function to create a table named 'employees'.
def create_table():
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS employees (
        id INTEGER PRIMARY KEY,
        name TEXT NOT NULL,
        age INTEGER,
        department TEXT
    )
    ''')
    conn.commit()
    conn.close()
    print("Table 'employees' created successfully.")

# Test the function
create_table()
```

This function creates a table named 'employees' with specified columns.

Assignment 2: Inserting Data

```
# Function to insert a new employee into the 'employees' table.
def insert_employee(id, name, age, department):
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    cursor.execute('''
    INSERT INTO employees (id, name, age, department)
    VALUES (?, ?, ?, ?)
    ''', (id, name, age, department))
    conn.commit()
```

```
    conn.close()
    print('Employee inserted successfully.')

# Insert 5 different employees
insert_employee(1, 'Alice', 30, 'HR')
insert_employee(2, 'Bob', 25, 'Engineering')
insert_employee(3, 'Charlie', 28, 'Sales')
insert_employee(4, 'David', 35, 'Marketing')
insert_employee(5, 'Eve', 22, 'HR')
```

This function inserts a new employee into the 'employees' table.

Assignment 3: Querying Data

```
# Function to fetch and display all records from the 'employees' table.
def fetch_all_employees():
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM employees')
    rows = cursor.fetchall()
    conn.close()
    for row in rows:
        print(row)

# Test the function
fetch_all_employees()
```

This function fetches and displays all records from the 'employees' table.

```
# Function to fetch and display all employees from a specific department.
def fetch_employees_by_department(department):
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM employees WHERE department = ?', (department,))
    records = cursor.fetchall()
    conn.close()
    for record in records:
        print(record)

# Test the function
fetch_employees_by_department('HR')
```

This function fetches and displays employees from a specified department.

Assignment 4: Updating Data

```
# Function to update the department of an employee based on their ID.
def update_employee_department(employee_id, new_department):
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    cursor.execute('''
```

```
    UPDATE employees
    SET department = ?
    WHERE id = ?
    ''', (new_department, employee_id))
    conn.commit()
    conn.close()
    print('Employee department updated successfully.')

# Test the function
update_employee_department(1, 'Finance')
```

This function updates the department of an employee based on their ID.

Assignment 5: Deleting Data

```
# Function to delete an employee from the 'employees' table based on their ID.
def delete_employee(employee_id):
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    cursor.execute('''
    DELETE FROM employees
    WHERE id = ?
    ''', (employee_id,))
    conn.commit()
    conn.close()
    print('Employee deleted successfully.')

# Test the function
delete_employee(5)
```

This function deletes an employee from the 'employees' table based on their ID.

Assignment 6: Advanced Queries

```
# Function to fetch and display employees older than a certain age.
def fetch_employees_older_age(age):
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
 cursor.execute('SELECT * FROM employees WHERE age > ?', (age,))
    records = cursor.fetchall()
    conn.close()
    for record in records:
        print(record)

# Test the function
fetch_employees_older_age(25)
```

This function fetches and displays employees older than a specified age.

```
# Function to fetch and display employees whose names start with a specific letter.
def fetch_employees_name_starts_with(letter):
```

```
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM employees WHERE name LIKE ?', (letter + '%',))
    records = cursor.fetchall()
    conn.close()
    for record in records:
        print(record)

# Test the function
fetch_employees_name_starts_with('A')
```

This function fetches and displays employees whose names start with a specified letter.

Assignment 7: Handling Transactions

```
# Function to insert multiple employees into the 'employees' table in a single transactio
def insert_multiple_employees(employees):
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    try:
        cursor.executemany('''
        INSERT INTO employees (id, name, age, department)
        VALUES (?, ?, ?, ?)
        ''', employees)
        conn.commit()
        print('All employees inserted successfully.')
    except Exception as e:
        conn.rollback()
        print('Error occurred, transaction rolled back.')
        print(e)
    finally:
        conn.close()

# Test the function with valid and invalid data
employees = [
    (6, 'Frank', 40, 'Finance'),
    (7, 'Grace', 29, 'Engineering'),
    (8, 'Hannah', 35, 'Marketing'),
    (9, 'Ivan', 38, 'Sales'),
    (6, 'Jack', 45, 'HR') # Duplicate ID to cause an error
]
insert_multiple_employees(employees)
```

This function inserts multiple employees in a single transaction, rolling back if any insertion fails.

```
# Function to update the ages of multiple employees in a single transaction.
def update_multiple_employees_ages(updates):
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    try:
        cursor.executemany('''
        UPDATE employees
```

```
        SET age = ?
        WHERE id = ?
        ''', updates)
        conn.commit()
        print('All employee ages updated successfully.')
    except Exception as e:
        conn.rollback()
        print('Error occurred, transaction rolled back.')
        print(e)
    finally:
        conn.close()

# Test the function with valid and invalid data
updates = [
    (32, 1),
    (26, 2),
    (33, 3),
    (41, 19), # Non-existing ID to cause an error
    (23, 5)
]
update_multiple_employees_ages(updates)
```

This function updates the ages of multiple employees in a single transaction, rolling back if any update fails.

Assignment 8: Creating Relationships

```
# Function to create a new table named 'departments'.
def create_departments_table():
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS departments (
        id INTEGER PRIMARY KEY,
        name TEXT NOT NULL
    )
    ''')
    conn.commit()
    conn.close()
    print("Table 'departments' created successfully.")

# Test the function
create_departments_table()
```

This function creates a new table named 'departments' with columns for ID and name.

```
# Function to modify the 'employees' table to include a foreign key referencing the 'depa
def add_department_foreign_key():
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()

    # Disable foreign keys temporarily
```

```
    cursor.execute("PRAGMA foreign_keys=off;")
    conn.commit()
    # Start transaction
    cursor.execute("BEGIN TRANSACTION;")
    # Rename old table
    cursor.execute("ALTER TABLE employees RENAME TO old_employees;")
    # Create new table with foreign key
    cursor.execute('''
    CREATE TABLE employees (
        id INTEGER PRIMARY KEY,
        name TEXT NOT NULL,
        age INTEGER,
        department TEXT,
        department_id INTEGER,
        FOREIGN KEY(department_id) REFERENCES departments(id)
    );
    ''')
    # Copy data, ensuring department_id is handled properly
    cursor.execute('''
    INSERT INTO employees (id, name, age, department, department_id)
    SELECT id, name, age, department, NULL FROM old_employees;
    ''')
    # Drop old table
    cursor.execute("DROP TABLE old_employees;")
    # Commit transaction
    conn.commit()
    # Re-enable foreign keys
    cursor.execute("PRAGMA foreign_keys=on;")
    conn.commit()
    conn.close()
    print("Table 'employees' modified successfully.")

# Test the function
add_department_foreign_key()
```

This function modifies the 'employees' table to include a foreign key that references the 'departments' table.

```
# Function to insert data into both the 'departments' and 'employees' tables, ensuring re
def insert_department_and_employee(department_id, department_name, employee_id, name, age
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    try:
        cursor.execute('''
        INSERT INTO departments (id, name)
        VALUES (?, ?)
        ''', (department_id, department_name))
        cursor.execute('''
        INSERT INTO employees (id, name, age, department, department_id)
        VALUES (?, ?, ?, ?, ?)
        ''', (employee_id, name, age, department, department_id))
        conn.commit()
        print("Department and employee inserted successfully.")
```

```
    except Exception as e:
        conn.rollback()
        print('Error occurred, transaction rolled back.')
        print(e)
    finally:
        conn.close()

# Test the function
insert_department_and_employee(1, 'HR', 10, 'Alice', 30, 'HR')
```

This function inserts data into both the 'departments' and 'employees' tables, ensuring referential integrity.

Assignment 9: Indexing and Optimization

```
# Function to create an index on the 'name' column of the 'employees' table.
def create_index_on_name():
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    cursor.execute('CREATE INDEX idx_name ON employees(name)')
    conn.commit()
    conn.close()
    print("Index on 'name' column created successfully.")

# Test the function
create_index_on_name()
```

This function creates an index on the 'name' column of the 'employees' table to optimize queries.

```
import time

def fetch_employees_name_starts_with_performance(letter):
    conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
    start_time = time.time()
    cursor.execute('SELECT * FROM employees WHERE name LIKE ?', (letter + '%',))
    records = cursor.fetchall()
    end_time = time.time()
    conn.close()
    print("Time taken: {} seconds".format(end_time - start_time))
    for record in records:
        print(record)

# Test the function with the index
fetch_employees_name_starts_with_performance('A')
```

This function fetches employees whose names start with a specified letter and measures the query performance.

Assignment 10: Backing Up and Restoring

```
# Function to back up the 'test.db' database to a file named 'backup.db'.
import shutil

def backup_database():
    shutil.copy('test.db', 'backup.db')
    print("Database backed up successfully.")

# Test the function
backup_database()
```

This function backs up the 'test.db' database to a file named 'backup.db'.

```
# Function to restore the 'test.db' database from the backup file 'backup.db'.
def restore_database():
    shutil.copy('backup.db', 'test.db')
    print("Database restored successfully.")

# Test the function
restore_database()
```

This function restores the 'test.db' database from the backup file 'backup.db'.